Reinforcement Learning from Static Datasets: Algorithms, Analysis, and Applications

Aviral Kumar

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2023-223
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2023/EECS-2023-223.html

August 11, 2023

Copyright © 2023, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

Reinforcement Learning from Static Datasets: Algorithms, Analysis, and Applications

By

Aviral Kumar

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Sergey Levine, Chair
Professor Jiantao Jiao
Professor Emma Brunskill
Professor Jennifer Listgarten

Summer 2023

Reinforcement Learning from Static Datasets:
Algorithms, Analysis, and Applications

Copyright © 2023

by

Aviral Kumar

Abstract

Reinforcement Learning from Static Datasets:
Algorithms, Analysis, and Applications

by

Aviral Kumar

Doctor of Philosophy in Computer Science

University of California, Berkeley

Professor Sergey Levine, Chair

Reinforcement learning (RL) provides a formalism for learning-based control. By attempting to learn behavioral policies that can optimize a user-specified reward function, RL methods have been able to acquire novel decision-making strategies that can outperform the best humans even with highly complex dynamics and even when the space of all possible outcomes is huge (e.g., robotic manipulation, chip floorplanning). Yet RL has had a limited applicability compared to standard machine learning (ML) in real-world scenarios. Why? The central issue with RL is that it relies crucially on running large amounts of trial-and-error active data collection for learning policies. Unfortunately though, in the real world, active data collection is generally very expensive (e.g., running wet lab experiments for drug design), and/or dangerous (e.g., robots operating around humans), and accurate simulators are hard to build. Overall, this means that while RL carries the potential to broadly unlock ML in real-world decision-making problems, we are unable to realize this potential via current RL techniques.

To realize this potential of RL, in this dissertation, we develop an alternate paradigm that aims to utilizes static datasets of experience for learning policies. Such a "dataset-driven" paradigm broadens the applicability of RL to a variety of decision-making problems where historical datasets already exist or can be collected via domain-specific strategies. It also brings the scalability and reliability benefits that modern supervised and unsupervised ML methods enjoy into RL. That said, instantiating this paradigm is challenging as it requires reconciling the static nature of learning from a dataset with the traditionally active nature of RL, which results in challenges of distributional shift, generalization, and optimization. After theoretically and empirically understanding these challenges, we develop algorithmic ideas for addressing thee challenges and discuss several extensions to convert these ideas into practical methods that can train modern high-capacity neural

network function approximators on large and diverse datasets. Finally, we show how the techniques can enable us to pre-train generalist policies for real robots and video games and enable fast and efficient hardware accelerator design.

CONTENTS

Acknowledgments

First and foremost, I am tremendously grateful to my adviser, Sergey Levine, for his continuous support and guidance throughout my Ph.D. Besides heavily influencing my technical growth and a lot of ideas in this dissertation, Sergey provided me with the freedom to work on problems of my interest, allowing me to pursue research that I wanted, and has taught me the art of asking the right questions, the importance of which I now realize. I am also grateful to Sergey for teaching me how to think about the big-picture questions. He is the smartest researcher I have met so far and I believe I could not have had a better advisor.

I am very grateful to George Tucker, who has been a long-term collaborator of mine since my second paper in my Ph.D. Most notably, I have learned from him the importance of critical examination of research ideas. George's critical questions have always made me and my work better and I really appreciate him for this. Thank you to my qualifying examination and dissertation committee members Jiantao Jiao, Emma Brunskill, and Jennifer Listgarten for your support, guidance, and fruitful discussions. I am also extremely grateful to Geoffrey Hinton for taking a chance on me when I did not know much about machine learning and hosting me as an intern at Google Brain Toronto in 2017, and for introducing me to reinforcement learning, which is the area I studied for my Ph.D. I am also very thankful to Pieter Abbeel for his advice on research and career at multiple points during my Ph.D., most notably during the first year of my PhD.

I have had the great fortune to collaborate with a number of faculty members, students, post-doctoral scholars, and researchers throughout my Ph.D. In the initial years of my Ph.D., I started exploring the topic of offline RL with Justin Fu. At the time, I also worked with Abhishek Gupta on several topics aimed at understanding temporal-difference learning. These were both immense learning experiences for me. I also learned a lot pertaining to the implementation of a variety of RL algorithms while working with Jason Peng. I am grateful to have had the opportunity to work with these senior graduate student collaborators in the initial years of my Ph.D.

Since the middle of my Ph.D., I have collaborated with and learned a lot from Chelsea Finn, Tianhe Yu, Rishabh Agarwal, Young Geng, Anikait Singh, Amir Yazdanbakhsh, Abhishek Gupta, Dibya Ghosh, Yi Su, Avi Singh, Frederik Ebert, Colin Li, and Quan Vuong. In particular, I would like to thank Rishabh Agarwal and Tianhe Yu, with whom I explored several new research topics that established a line of work covered in this thesis. I would like to thank Rishabh Agarwal for teaching me how to systematically organize experiments and compile results, a skill that will be very valuable for my future. I would like to thank Young Geng for answering countless queries about Jax, being my go-to person to ask for implementation advice, and asking hard questions on projects I pursued and my research ideas that enabled me to grow as a researcher. I am also grateful

to all other collaborators I have worked with: Yi Su, Mitsuhiko Nakamoto, Saurabh Kumar, Simon Zhai, Anurag Ajay, Aurick Zhou, Joey Hong, Tengyu Ma, Karol Hausman, Yevgen Chebotar, Zhang-Wei Hong, Manan Tomar, Katie Kang, Kuba Grudzien, Nick Rhinehart, Rafael Rafailov, Amy Lu, Ofir Nachum, Aravind Rajeswaran, Homer Walke, Florian Shkurti, Animesh Garg, Nilah Ionidis, Grace Gu, Zico Kolter, Priya Donti, Melrose Roderick, Jad Rahme, Milad Hashemi, Will Chen, Yifei Zhou, and Pulkit Agarwal. I would also especially like to thank all undergraduate and master's students I worked with: Brandon Trabucco, Anikait Singh, Kevin Li, Yanlai Yang, Sathvik Kolli, Homanga Bharadhwaj, Jonathan Yang, Albert Yu, Stephen Tian, Han Qi, Ria Doshi, Chet Bhateja, Derek Guo, and Jason Wang for their hard work and dedication to the projects. These students are now leaders in emerging areas of AI and I eagerly look forward to their future successes.

During my Ph.D., I was a part-time student researcher at Google Brain from 2020 to 2023. I was hosted by George Tucker and Amir Yazdanbakhsh, two excellent mentors, who gave me the freedom to work on any topic I wanted and collaborate with anyone. I started to appreciate the importance of looking at applied problems and learned a lot about systems working with Amir. Amir would never turn my silly questions time and always make time to discuss with me, which I am very thankful about. I am grateful to all researchers and engineers I have talked to and collaborated with at Google during this time. Many times a lot of these discussions and collaborations never made it into a publication or release, but gave me a lot of perspective about real-world problems. Thank you, Sandra Faust, Dale Schuurmans, Minmin Chen, Marc Bellemare, Yevgen Chebotar, Karol Hausman, Hossein Mobahi, Kevin Swersky, Mohammad Norouzi, Yundi Qian, John Sipple, Sherry Yang, Yingjie Miao, Jordi Orbay, and Ofir Nachum, for sharing your perspectives and thoughtful conversations.

I would also like to thank the members of the RAIL lab and the broader BAIR community for creating a supportive and friendly atmosphere in the lab. Thanks to the open layout of the lab, I enjoyed engaging in several conversations on BWW8, that would start randomly over lunch, coffee, or near the desk area. Special thanks to Young Geng and Michael Janner, graduate students in RAIL in my cohort: our high-level philosophical discussions about research trends over coffee and food have greatly shaped my thoughts. The middle years of my Ph.D. were during the height of the COVID pandemic and hence, a time when we were all remote, but I greatly enjoyed hanging out with and learned a lot from the members of RAIL before and after. Thank you, Michael Janner, Abhishek, Frederik, Justin, Michael Chang, Aurick, Kelvin, Nick, Rowan, Natasha, Dhruv, Manan, Dibya, Colin, Laura, Katie, Marwa, Amy Lu, Mitsuhiko, Simon, Oleh, Karl, Homer, Kuba, Kevin, Kuan, Amy Zhang, Natasha, Seohong, Will, Kyle, Kristian, Vivek, Philip Ball, Glen, Anusha, Coline, Greg, Dinesh, Vitchyr, JD, Jason, Sid, Avi, Ashvin, Alex Lee, Erin, Marvin, Joey, Charlie, Tuomas, Kate, Jedrzej, and others, for making my Ph.D. years exciting, both

research and otherwise. I especially cherish my experiences with hiking "buddies" in RAIL including Manan, Colin, and Young, who over a period of 2 months made me realize how beautiful Berkeley hills are and transformed me from someone who would not at all want to walk up Hearst to someone who loves to go up to the peaks in Berkeley hills everyday. I also am grateful to all dinner conversations I have had with Young, Oleh, Karl, and Mitsuhiko, all of which have only made me a better person. Also big thanks to lab members nearby: Young Geng, Marwa Abdulhai, Laura Smith, Rowan McAllister, Amy Zhang, Abhishek Gupta, and Marvin Zhang, for making it fun to be in the lab.

My experience during graduate school has been made ever so enjoyable by my amazing group of friends outside work, both in Berkeley and elsewhere. Thank you, Karan, Siddharth, Shubham, Anurag, and Manan, for being extraordinary roommates and for supporting and helping me selflessly whenever I asked. Thank you, Saurabh, Siddhant, Sumith, Amrith, Utkarsha, Moraldeep, and Upadhi, for providing me fundae at various points during my PhD years. I want to thank Jean Nguyen, Angie Abbatecola, and the rest of the BAIR admin team for their help in making my Ph.D. experience smooth and enjoyable. My time at Berkeley would also not have been the same without the coffee from Blue Bottle and tea from Asha's, the deadline food from the Noodle, Bangkok Thai, Mad Seoul, Bongo Burger, and Urban Turbann, as well as 6 am paper deadline breakfasts from Noah's Bagels and McDonald's.

Finally, I would like to thank my parents, Mukesh Saxena and Sadhna Sinha: thank you for giving me the freedom to pursue my dreams, keeping me sane in times of stress, and never letting me feel alone from the other side of the planet.

1

INTRODUCTION

Reinforcement learning (RL) provides a mathematical formalism for learning-based control. Specifically, reinforcement learning techniques can automatically discover and acquire near-optimal behavior (often referred to as policies), which optimizes a user-specified reward function. The reward function defines *what* a policy should do, and an RL algorithm automatically determines *how* to do it. Devising RL algorithms has been an active area of research ever since the development of dynamic programming algorithms for optimal control in the 1960s [25]. Over the last decade, the introduction of effective high-capacity deep neural network function approximators into RL, along with effective algorithms for training them, has allowed RL methods to attain excellent results along a wide range of domains [321, 202, 232, 203, 298, 160], often producing policies that match or outperform the best known control strategy for the downstream task.

Despite these promising results, reinforcement learning methods have had limited applicability in a number of other decision-making and control problems in the real world. This is because traditional reinforcement learning techniques that have been developed for a few decades now typically subscribe to an *online* learning paradigm: these techniques involve iteratively collecting experience by actively interacting with an environment in a trial-and-error fashion, and then using that experience to improve the policy [312]. In many settings, this sort of online interaction is impractical, either because data collection is expensive (e.g., in robotics, drug design, education, power grid management, or healthcare) and dangerous (e.g., in autonomous driving, power grid management, or healthcare). One option to circumvent the need for active interaction is to instead build simulators for the target problem and then run RL in simulation for discovering the optimal behaviors. Even though this approach is practically feasible, it tends to perform poorly when accurate simulators are hard to build (e.g., modelling the interaction between a drug and a pathogen or that between humans and robots).

An alternate approach that we take in this dissertation is to **build a "dataset-driven" paradigm for RL that can incorporate static, previously-collected datasets for making effective decisions**. Since the success of supervised and unsupervised machine learning

methods across a range of practically relevant problems over the past decade can in large part be attributed to the development of scalable dataset-driven learning methods, which continue to reliably improve as they are trained with more and higher-quality data, one would also expect that such a dataset-driven paradigm for RL will enjoy these appealing properties. Put in other words, such a dataset-driven paradigm for RL bears the promise of translating progress in collecting datasets and advances in deep learning, directly to generalizable and powerful decision-making engines.

That said, instantiating this paradigm presents a challenge as it requires reconciling the **passive** nature of a static dataset with traditional RL techniques and objective functions, which critically rely on **actively** collecting experience for improving the underlying policy. While prior work shows that the class of off-policy RL algorithms, which we briefly review in Chapter 2, can suffice for this fully offline setting as well [192, 43, 234, 235], these classical techniques often require restrictive assumptions that do not hold with high-dimensional control problems and realistic datasets. As we will discuss in Chapter 3, in realistic decision-making problems, the inability to actively collect experience often manifests in the form of several challenges pertaining to distributional shift, generalization, and optimization. These challenges are perhaps the most evident in the fully *"offline"* setting, where we are not allowed access to any form of active interaction. Function approximators such as deep neural networks generally exacerbate these issues, since function approximation increases the vulnerability of the learning algorithm to pathologies from distributional shift and incorrect generalization.

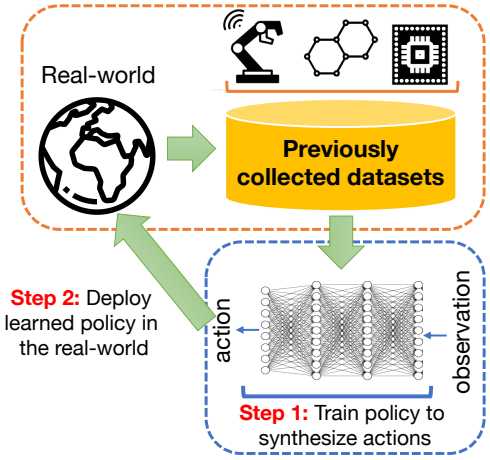


Figure 1: The offline RL paradigm.

Concretely, in this dissertation we provide a theoretical and empirical foundation to the problem of utilizing static datasets in reinforcement learning, especially relevant within the context of realistic datasets. We perform empirical studies to isolate the various challenges when learning policies via RL in a fully offline manner, followed by developing algorithmic techniques to address these challenges. Then, we develop techniques to scale up these algorithmic ideas to leverage the generalization benefits offered by highly-expressive function approximators, especially when provided with diverse and multi-task datasets from a rich set of RL problems. We also develop techniques that enable rapid fine-tuning of an offline initialization learned from the static dataset, with a limited amount of active interaction. Finally, we demonstrate the efficacy of these algorithmic ideas in the context of two real-world decision-making problems in hardware accelerator design (i.e., chip design) and real robot control.

Organization. After a brief disscussion of notation and the problem statement in Chapter 2 (based on Levine et al. [204]), we make the following contributions:

- In Chapter 3, we empirically analyze the behavior of traditional RL algorithms in the offline learning setting and isolate the challenges of distributional shift and overfitting. This work appeared previously as Fu et al. [90].
- In Chapter 4, we develop an initial algorithm to circumvent the challenge of distributional shift by restricting the learned policy to lie within the *support* of the data-collection policy. This work was published previously as Kumar et al. [179].
- In Chapter 5, we develop an approach for learning policies from static data, that we call conservative value estimation. Conservative estimation of value functions dispenses with the need to restrict the learned policy to within the support of the data-collection policy, enabling bigger performance improvements, while still addressing the challenge of distributional shift. We also instantiate our approach into concrete model-free and model-based algorithms. This chapter consists of content from work previously published as Kumar et al. [181], Yu et al. [367].
- In Chapter 6, we develop explicit regularization techniques that enable the approach from Chapter 5 to utilize high-capacity neural network architectures. This chapter is based on works previously published as Kumar et al. [183, 185].
- In Chapter 7, we develop automatic data sharing and reward labeling strategies that boost the performance of the approach from Chapter 5, when learning from multi-task and unlabeled datasets. This chapter consists of work that was published as Yu et al. [366, 368].
- In Chapter 8, we develop a method that enables sample-efficient online fine-tuning of offline policy initializations learned by the approach in Chapter 5. This chapter is primarily based on work that previously appears as Nakamoto et al. [244].
- In Chapter 9, we present an application of the techniques from Chapters 5, 6, and 8 to the problem of incorporating broad prior data for boosting generalization of robotic skill learning, with only a handful of task-specific rollouts. This chapter consists of work that was previously published as Kumar et al. [188].
- In Chapter 10, we present an application of the techniques from Chapter 5 to the problem of hardware accelerator design. This chapter consists of work that was published as Kumar et al. [177].

We conclude with a discussion of current approaches and promising future directions in developing reinforcement learning algorithms that can incorporate static datasets.

Part I

PROBLEM STATEMENT AND CHALLENGES

2

PROBLEM STATEMENT AND PRELIMINARIES

Abstract

In this chapter, we will discuss our notion and background definition, followed by a discussion of the problem statement of offline reinforcement learning (RL).

2.1 REINFORCEMENT LEARNING PRELIMINARIES

In this section, we will define reinforcement learning (RL) concepts, following standard definitions [314]. RL addresses the problem of learning to control a dynamical system. The dynamical system is fully defined by a fully-observed or partially-observed Markov decision process (MDP). We only consider problems where the dnynamical system is defined by a fully-observed MDP in this dissertation.

Definition 2.1.1 (Markov decision process)**.** *A Markov decision process is defined as a tuple* $\mathcal{M} = (\mathcal{S}, \mathcal{A}, T, d_0, r, \gamma)$*, where* $\mathcal{S}$ *is a set of states* $\boldsymbol{s} \in \mathcal{S}$*, which may be either discrete or continuous (i.e., multi-dimensional vectors),* $\mathcal{A}$ *is a set of actions* $\boldsymbol{a} \in \mathcal{A}$*, which similarly can be discrete or continuous,* T *defines a conditional probability distribution of the form* $T(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)$ *that describes the dynamics of the system,*[1] d_0 *defines the initial state distribution* $d_0(\boldsymbol{s}_0)$*,* $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ *defines a reward function, and* $\gamma \in (0, 1]$ *is a scalar discount factor.*

The final goal in a reinforcement learning problem is to learn a policy, which defines a distribution over actions conditioned on states, $\pi(\boldsymbol{a}_t|\boldsymbol{s}_t)$. From these definitions, we can derive the *trajectory distribution*. The trajectory is a sequence of states and actions

[1] We will sometimes use time subscripts (i.e., $\boldsymbol{s}_{t+1}$ follows $\boldsymbol{s}_t$), and sometimes "prime" notation (i.e., $\boldsymbol{s}'$ is the state that follows $\boldsymbol{s}$). Explicit time subscripts can help clarify the notation in finite-horizon settings, while "prime" notation is simpler in infinite-horizon settings where absolute time step indices are less meaningful.

of length H, given by $\tau = (\boldsymbol{s}_0, \boldsymbol{a}_0, \ldots, \boldsymbol{s}_H, \boldsymbol{a}_H)$, where H may be infinite. The trajectory distribution p_π for a given MDP $\mathcal{M}$ and policy π is given by

$$p_\pi(\tau) = d_0(\boldsymbol{s}_0) \prod_{t=0}^{H} \pi(\boldsymbol{a}_t|\boldsymbol{s}_t) T(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t).$$

The reinforcement learning objective, $J(\pi)$, can then be written as an expectation under this trajectory distribution:

$$J(\pi) = \mathbb{E}_{\tau \sim p_\pi(\tau)} \left[\sum_{t=0}^{H} \gamma^t r(\boldsymbol{s}_t, \boldsymbol{a}_t) \right]. \tag{2.1.1}$$

When H is infinite, it is typical to consider the expected reward under the γ-discounted stationary distribution of the learned policy. Formally, we can refer to the marginals of the trajectory distribution $p_\pi(\tau)$. We will use $d^\pi(\boldsymbol{s})$ to refer to the overall state visitation frequency, averaged over the time steps, and $d_t^\pi(\boldsymbol{s}_t)$ to refer to the state visitation frequency at time step t. Alternatively, we can consider the undiscounted expected reward under the stationary distribution of the Markov chain $(\boldsymbol{s}_t, \boldsymbol{a}_t)$ defined by $\pi(\boldsymbol{a}_t|\boldsymbol{s}_t)T(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)$, under ergodicity assumptions. For discussions in this dissertation, we will consider the discounted marginal setting for simplicity.

Next, we will briefly summarize some types reinforcement learning algorithms and present definitions. At a high level, all standard reinforcement learning algorithms follow the same basic learning loop: the agent *interacts* with the MDP $\mathcal{M}$ by using some sort of *behavior policy*, which may or may not match $\pi(\boldsymbol{a}|\boldsymbol{s})$, by observing the current state $\boldsymbol{s}_t$, selecting an action $\boldsymbol{a}_t$, and then observing the resulting next state $\boldsymbol{s}_{t+1}$ and reward value $r_t = r(\boldsymbol{s}_t, \boldsymbol{a}_t)$. This may repeat for multiple steps, and the agent then uses the observed transitions $(\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t+1}, r_t)$ to update its policy. This update might also utilize previously observed transitions. We will use $\mathcal{D} = \{(\boldsymbol{s}_t^i, \boldsymbol{a}_t^i, \boldsymbol{s}_{t+1}^i, r_t^i)\}$ to denote the set of transitions that are available for the agent to use for updating the policy ("learning"), which may consist of either all transitions seen so far, or some subset thereof.

Dynamic Programming with Function Approximators

One way to optimize the reinforcmeent learning objective relies on the following observation: if we can accurately estimate a state or state-action *value function*, then it is easy to then recover a near-optimal policy. A value function provides an estimate of the expected cumulative reward that will be obtained by following some policy $\pi(\boldsymbol{a}_t|\boldsymbol{s}_t)$ when starting from a given state $\boldsymbol{s}_t$, in the case of the state-value function $V^\pi(\boldsymbol{s}_t)$, or when starting from

a state-action tuple $(\boldsymbol{s}_t, \boldsymbol{a}_t)$, in the case of the state-action value function $Q^\pi(\boldsymbol{s}_t, \boldsymbol{a}_t)$. We can define these value functions as:

$$V^\pi(\boldsymbol{s}_t) = \mathbb{E}_{\tau \sim p_\pi(\tau|\boldsymbol{s}_t)} \left[\sum_{t'=t}^{H} \gamma^{t'-t} r(\boldsymbol{s}_t, \boldsymbol{a}_t) \right]$$
$$Q^\pi(\boldsymbol{s}_t, \boldsymbol{a}_t) = \mathbb{E}_{\tau \sim p_\pi(\tau|\boldsymbol{s}_t, \boldsymbol{a}_t)} \left[\sum_{t'=t}^{H} \gamma^{t'-t} r(\boldsymbol{s}_t, \boldsymbol{a}_t) \right].$$

From this, we can derive recursive definitions for value functions:

$$V^\pi(\boldsymbol{s}_t) = \mathbb{E}_{\boldsymbol{a}_t \sim \pi(\boldsymbol{a}_t|\boldsymbol{s}_t)} \left[Q^\pi(\boldsymbol{s}_t, \boldsymbol{a}_t) \right]$$
$$Q^\pi(\boldsymbol{s}_t, \boldsymbol{a}_t) = r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma \mathbb{E}_{\boldsymbol{s}_{t+1} \sim T(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)} \left[V^\pi(\boldsymbol{s}_{t+1}) \right].$$

We can combine these two equations to express the $Q^\pi(\boldsymbol{s}_t, \boldsymbol{a}_t)$ in terms of $Q^\pi(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1})$:

$$Q^\pi(\boldsymbol{s}_t, \boldsymbol{a}_t) = r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma \mathbb{E}_{\boldsymbol{s}_{t+1} \sim T(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t), \boldsymbol{a}_{t+1} \sim \pi(\boldsymbol{a}_{t+1}|\boldsymbol{s}_{t+1})} \left[Q^\pi(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}) \right]. \tag{2.1.2}$$

We can also express these in terms of the *Bellman operator* for the policy π, which we denote $\mathcal{B}^\pi$. For example, Equation (2.1.2) can be written as $Q^\pi = \mathcal{B}^\pi Q^\pi$, where Q^π (with abuse of notation) now denotes the Q-function Q^π represented as a vector of length $|\mathcal{S}| \times |\mathcal{A}|$. Before moving on to deriving learning algorithms based on these definitions, we briefly discuss some properties of the Bellman operator. This Bellman operator has a unique fixed point that corresponds to the true Q-function for the policy $\pi(\boldsymbol{a}|\boldsymbol{s})$, which can be obtained by repeating the iteration $Q^\pi_{k+1} = \mathcal{B}^\pi Q^\pi_k$, and it can be shown that $\lim_{k \to \infty} Q^\pi_k = Q^\pi$, which obeys Equation (2.1.2) [314]. The proof for this follows from the observation that $\mathcal{B}^\pi$ is a contraction in the ℓ_∞ norm [191].

Based on these definitions, we can derive two commonly used algorithms based on dynamic programming: Q-learning and actor-critic methods. To derive Q-learning, we express the policy implicitly in terms of Q as $\pi(\boldsymbol{a}_t|\boldsymbol{s}_t) = \delta(\boldsymbol{a}_t = \arg\max Q(\boldsymbol{s}_t, \boldsymbol{a}_t))$, and only learn the Q-function $Q(\boldsymbol{s}_t, \boldsymbol{a}_t)$. By substituting this (implicit) policy into the above dynamic programming equation, we obtain the following condition on the optimal Q-function:

$$Q^\star(\boldsymbol{s}_t, \boldsymbol{a}_t) = r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma \mathbb{E}_{\boldsymbol{s}_{t+1} \sim T(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)} \left[\max_{\boldsymbol{a}_{t+1}} Q^\star(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}) \right]. \tag{2.1.3}$$

We can again express this as $Q = \mathcal{B}^\star Q$ in vector notation, where $\mathcal{B}^\star$ now refers to the Bellman optimality operator. Note however that this operator is not linear, due to the maximization on the right-hand side in Equation (2.1.3). To turn this equation into a learning algorithm, we can minimize the difference between the left-hand side and right-hand side of this equation with respect to the parameters of a parametric Q-function

estimator with parameters ϕ, $Q_\phi(\boldsymbol{s}_t, \boldsymbol{a}_t)$. There are a number of variants of this Q-learning procedure, including variants that fully minimize the difference between the left-hand side and right-hand side of the above equation at each iteration, commonly referred to as fitted Q-iteration [72, 277], and variants that take a single gradient step, such as the original Q-learning method [337]. The commonly used variant in deep reinforcement learning is a kind of hybrid of these two methods, employing a replay buffer [212] and taking gradient steps on the Bellman error objective concurrently with data collection [232]. We write out a general recipe for Q-learning methods in Algorithm 1.

Algorithm 1 Generic Q-learning (includes FQI and DQN as special cases)

initialize ϕ_0
initialize $\pi_0(\boldsymbol{a}|\boldsymbol{s}) = \epsilon\mathcal{U}(\boldsymbol{a}) + (1-\epsilon)\delta(\boldsymbol{a} = \arg\max_{\boldsymbol{a}} Q_{\phi_0}(\boldsymbol{s}, \boldsymbol{a}))$ ▷ Use ϵ-greedy exploration
initialize replay buffer $\mathcal{D} = \emptyset$ as a ring buffer of fixed size
initialize $\boldsymbol{s} \sim d_0(\boldsymbol{s})$
for iteration $k \in [0, \ldots, K]$ **do**
    for step $s \in [0, \ldots, S-1]$ **do**
        $\boldsymbol{a} \sim \pi_k(\boldsymbol{a}|\boldsymbol{s})$ ▷ sample action from exploration policy
        $\boldsymbol{s}' \sim p(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a})$ ▷ sample next state from MDP
        $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}', r(\boldsymbol{s}, \boldsymbol{a}))\}$ ▷ append to buffer, purging old data if buffer too big
    end for
    $\phi_{k,0} \leftarrow \phi_k$
    for gradient step $g \in [0, \ldots, G-1]$ **do**
        sample batch $batch \subset \mathcal{D}$ ▷ $B = \{(\boldsymbol{s}_i, \boldsymbol{a}_i, \boldsymbol{s}'_i, r_t)\}$
        estimate error $\mathcal{E}(B, \phi_{k,g}) = \sum_i \left(Q_{\phi_{k,g}} - (r_i + \gamma \max_{\boldsymbol{a}'} Q_{\phi_k}(\boldsymbol{s}', \boldsymbol{a}'))\right)^2$
        update parameters: $\phi_{k,g+1} \leftarrow \phi_{k,g} - \alpha\nabla_{\phi_{k,g}}\mathcal{E}(B, \phi_{k,g})$
    end for
    $\phi_{k+1} \leftarrow \phi_{k,G}$ ▷ update parameters
end for

Classic Q-learning can be derived as the limiting case where the buffer size is 1, and we take $G = 1$ gradient steps and collect $S = 1$ transition samples per iteration, while classic fitted Q-iteration runs the inner gradient descent phase to convergence (i.e., $G = \infty$), and uses a buffer size equal to the number of sampling steps S. Note that many modern implementations also employ a *target network*, where the target value $r_i + \gamma \max_{\boldsymbol{a}'} Q_{\phi_k}(\boldsymbol{s}', \boldsymbol{a}')$ actually uses ϕ_L, where L is a lagged iteration (e.g., the last k that is a multiple of 1000). Note that these approximations violate the assumptions under which Q-learning algorithms can be proven to converge. However, recent work suggests that high-capacity function approximators, which correspond to a very large set

$\mathcal{Q}$, generally do tend to make this method convergent in practice, yielding a Q-function that is close to Q^π [90, 329].

Algorithm 2 Generic off-policy actor-critic

initialize ϕ_0
initialize θ_0
initialize replay buffer $\mathcal{D} = \emptyset$ as a ring buffer of fixed size
initialize $\boldsymbol{s} \sim d_0(\boldsymbol{s})$
for iteration $k \in [0, \ldots, K]$ **do**
    for step $s \in [0, \ldots, S-1]$ **do**
        $\boldsymbol{a} \sim \pi_{\theta_k}(\boldsymbol{a}|\boldsymbol{s})$ ▷ sample action from current policy
        $\boldsymbol{s}' \sim p(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a})$ ▷ sample next state from MDP
        $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}', r(\boldsymbol{s}, \boldsymbol{a}))\}$ ▷ append to buffer, purging old data if buffer too big
    end for
    $\phi_{k,0} \leftarrow \phi_k$
    for gradient step $g \in [0, \ldots, G_Q - 1]$ **do**
        sample batch $batch \subset \mathcal{D}$ ▷ $B = \{(\boldsymbol{s}_i, \boldsymbol{a}_i, \boldsymbol{s}'_i, r_t)\}$
        estimate error $\mathcal{E}(B, \phi_{k,g}) = \sum_i \left(Q_{\phi_{k,g}} - (r_i + \gamma \mathbb{E}_{\boldsymbol{a}' \sim \pi_k(\boldsymbol{a}'|\boldsymbol{s}')} Q_{\phi_k}(\boldsymbol{s}', \boldsymbol{a}')) \right)^2$
        update parameters: $\phi_{k,g+1} \leftarrow \phi_{k,g} - \alpha_Q \nabla_{\phi_{k,g}} \mathcal{E}(B, \phi_{k,g})$
    end for
    $\phi_{k+1} \leftarrow \phi_{k,G_Q}$ ▷ update Q-function parameters
    $\theta_{k,0} \leftarrow \theta_k$
    for gradient step $g \in [0, \ldots G_\pi - 1]$ **do**
        sample batch of states $\{\boldsymbol{s}_i\}$ from $\mathcal{D}$
        for each $\boldsymbol{s}_i$, sample $\boldsymbol{a}_i \sim \pi_{\theta_{k,g}}(\boldsymbol{a}|\boldsymbol{s}_i)$ ▷ do not use actions in the buffer!
        for each $(\boldsymbol{s}_i, \boldsymbol{a}_i)$, compute $\hat{A}(\boldsymbol{s}_i, \boldsymbol{a}_i) = Q_{\phi_{k+1}}(\boldsymbol{s}_i, \boldsymbol{a}_i) - \mathbb{E}_{\boldsymbol{a} \sim \pi_{k,g}(\boldsymbol{a}|\boldsymbol{s}_i)}[Q_{\phi_{k+1}}(\boldsymbol{s}_i, \boldsymbol{a})]$
        $\nabla_{\theta_{k,g}} J(\pi_{\theta_{k,g}}) \approx \frac{1}{N} \nabla_{\theta_{k,g}} \log \pi_{\theta_{k,g}}(\boldsymbol{s}_i, \boldsymbol{a}_i) \hat{A}(\boldsymbol{s}_i, \boldsymbol{a}_i)$
        $\theta_{k,g+1} \leftarrow \theta_{k,g} + \alpha_\pi \nabla_{\theta_{k,g}} J(\pi_{\theta_{k,g}})$
    end for
    $\theta k + 1 \leftarrow \theta_{k,G_\pi}$ ▷ update policy parameters
end for

Actor-Critic Algorithms

Actor-critic algorithms employ *both* a parameterized policy and a parameterized value function, and use the value function to provide training signal for the policy. Typically, the learned value function is used to provide a better estimate of policy performance (or

advantage $\hat{A}(\boldsymbol{s}, \boldsymbol{a})$) in a policy gradient objective. There are a number of different variants of actor-critic methods, including on-policy variants that directly estimate $V^{\pi}(\boldsymbol{s})$ [171], and off-policy variants that estimate $Q^{\pi}(\boldsymbol{s}, \boldsymbol{a})$ via a parameterized state-action value function $Q^{\pi}_{\phi}(\boldsymbol{s}, \boldsymbol{a})$ [125, 124, 134].

We will focus on the latter class of algorithms, since they can be easily extended to the offline setting. The basic design of such an algorithm is a straightforward combination of the ideas in dynamic programming and policy gradients. Unlike Q-learning, which directly attempts to learn the optimal Q-function, actor-critic methods aim to learn the Q-function corresponding to the current parameterized policy $\pi_{\theta}(\boldsymbol{a}|\boldsymbol{s})$, which must obey the equation

$$Q^{\pi}(\boldsymbol{s}_t, \boldsymbol{a}_t) = r(\boldsymbol{s}_t, \boldsymbol{a}_t) + \gamma \mathbb{E}_{\boldsymbol{s}_{t+1} \sim T(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t), \boldsymbol{a}_{t+1} \sim \pi_{\theta}(\boldsymbol{a}_{t+1}|\boldsymbol{s}_{t+1})} \left[Q^{\pi}(\boldsymbol{s}_{t+1}, \boldsymbol{a}_{t+1}) \right].$$

As before, this equation can be expressed in vector form in terms of the Bellman operator for the policy, $Q^{\pi} = \mathcal{B}^{\pi} Q^{\pi}$, where Q^{π} denotes the Q-function Q^{π} represented as a vector of length $|\mathcal{S}| \times |\mathcal{A}|$. We can now instantiate a complete algorithm based on this idea, shown in Algorithm 2.

For more details, we refer the reader to standard textbooks and prior works [312, 171]. Actor-critic algorithms are closely related with another class of methods that frequently arises in dynamic programming, called policy iteration (PI) [191]. Policy iteration consists of two phases: policy evaluation and policy improvement. The policy evaluation phase computes the Q-function for the current policy π, Q^{π}, by solving for the fixed point such that $Q^{\pi} = \mathcal{B}^{\pi} Q^{\pi}$. This can be done via gradient updates, analogously to line 15 in Algorithm 2. The next policy iterate is then computed in the policy improvement phase, by choosing the action that greedily maximizes the Q-value at each state, such that $\pi_{k+1}(\boldsymbol{a}|\boldsymbol{s}) = \delta(\boldsymbol{a} = \arg\max_{\boldsymbol{a}} Q^{\pi_k}(\boldsymbol{s}, \boldsymbol{a}))$, or by using a gradient based update procedure as is employed in Algorithm 2 on line 24. In the absence of function approximation (e.g., with tabular representations) policy iteration produces a monotonically improving sequence of policies, and converges to the optimal policy. Policy iteration can be obtained as a special case of the generic actor-critic algorithm in Algorithm 2 when we set $G_Q = \infty$ and $G_{\pi} = \infty$, when the buffer $\mathcal{D}$ consists of each and every transition of the MDP.

Model-Based Reinforcement Learning

Model-based reinforcement learning is a general term that refers to a broad class of methods that utilize explicit estimates of the transition or dynamics function $T(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)$, parameterized by a parameter vector ψ, which we will denote $T_{\psi}(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)$. There is no single recipe for a model-based reinforcement learning method. Some commonly used model-based reinforcement learning algorithms learn only the dynamics model $T_{\psi}(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t)$, and then utilize it for planning at test time, often by means of model-predictive control (MPC) [318] with various trajectory optimization methods [239, 49].

Other model-based reinforcement learning methods utilize a learned policy $\pi_\theta(\boldsymbol{a}_t|\boldsymbol{s}_t)$ in addition to the dynamics model, and employ backpropagation through time to optimize the policy with respect to the expected reward objective [60]. Yet another set of algorithms employ the model to generate "synthetic" samples to augment the sample set available to model-free reinforcement learning methods. The classic Dyna algorithm uses this recipe in combination with Q-learning and one-step predictions via the model from previously seen states [313], while a variety of recently proposed algorithms employ synthetic model-based rollouts with policy gradients [257, 156] and actor-critic algorithms [148].

Comparison of different types of RL methods. One might wonder how these different classes of reinforcement learning methods compare with each other. In practice, value-based RL methods that use pproximate dynamic programming (Q-learning and actor-critic) train Q-functions to match non-stationary target values, resulting in an optimization problem different from standard supervised learning. This results in error propagation, a phenomenon that we discuss theoretically and empirically in Chapter 4 when learning value functions. On the other hand, training a model of the transition dynamics is a supervised learning problem, which can be tackled using well-studied tools in empirical risk minimization theoretically and benefits directly from advances in supervised deep learning in practice. That said, trajectory rollouts in model-based RL algorithms diverge from rollouts in the ground-truth model over longer horizons, once errors in fitting the model compound together over steps of a trajectory. This is further exacerbated when the policy aims to optimize the cumulative reward under the learned model, as we elaborate theoretically and empirically in Chapter 5. From a theoretical standpoint, value-based RL methods based on approximate dynamic programming require stronger assumptions of completeness in order to learn a policy effectively, although model-based RL methods do not require this sort of an assumption (see Sun et al. [310] for a detailed discussion of when model-based RL methods can perform better).

2.2 Problem Statement: Offline Reinforcement Learning

The offline reinforcement learning problem can be defined as a *dataset-driven* formulation of the RL problem. The end goal is still to optimize the objective in Equation (2.1.1). However, the agent no longer has the ability to interact with the environment and collect additional transitions using the behavior policy. Instead, the learning algorithm is provided with a *static* dataset of transitions, $\mathcal{D} = \{(\boldsymbol{s}^i_t, \boldsymbol{a}^i_t, \boldsymbol{s}^i_{t+1}, r^i_t)\}$, and must learn the best policy it can using this dataset. This formulation more closely resembles the standard supervised learning problem statement, and we can regard $\mathcal{D}$ as the *training set* for the policy. In essence, offline reinforcement learning requires the learning algorithm to derive a sufficient understanding of the dynamical system underlying the MDP $\mathcal{M}$ entirely from a fixed dataset, and then construct a policy $\pi(\boldsymbol{a}|\boldsymbol{s})$ that attains the largest

possible cumulative reward *when it is actually used to interact with the MDP*. We will use π_β to denote the distribution over states and actions in $\mathcal{D}$, such that we assume that the state-action tuples $(\boldsymbol{s}, \boldsymbol{a}) \in \mathcal{D}$ are sampled according to $\boldsymbol{s} \sim d^{\pi_\beta}(\boldsymbol{s})$, and the actions are sampled according to the behavior policy, such that $\boldsymbol{a} \sim \pi_\beta(\boldsymbol{a}|\boldsymbol{s})$.

The offline reinforcement learning problem can be approached using algorithms from many categories of examples discussed above, and in principle any off-policy RL algorithm *could* be used as an offline RL algorithm. For example, a simple offline RL method can be obtained simply by using Q-learning without additional online exploration, using $\mathcal{D}$ to pre-populate the data buffer. This corresponds to changing the initialization of $\mathcal{D}$ in Algorithm 1, and setting $S = 0$. However, as we will discuss in subsequent chapters, not all such methods are effective in the offline setting.

We will also consider an extension of the offline reinforcement learning problem, where the goal is to first learn an offline policy initialization, followed by fine-tuning the learned policy with limited amounts of online, actively-collected data. We will define a bit of terminology and notation for this problem in Chapter 8, which studies this problem.

3

CHALLENGES IN OFFLINE REINFORCEMENT LEARNING

Abstract

In this chapter, we aim to understand challenges in learning policie from offline data. To this end, we empirically study the behavior of a class of standard RL methods when training on a static, offline dataset. In principle, off-policy RL methods should be able to leverage experience collected from prior policies for sample-efficient learning. However, as we illustrate in this chapter, in practice, commonly used off-policy RL methods from Chapter 2 based on Q-learning and actor-critic are incredibly sensitive to *both* the dataset distribution and quantity. Building on these insights from our empirical study, we identify two main challenges in the offline setting: *distributional shift* and *sampling error*. In the subsequent chapters, we develop techniques to handle distributional shift (Chapters 4, 5) and briefly, sampling error (Chapter 6), progressing towards reliable and easy-to-use offline RL methods.

3.1 INTRODUCTION

In principle, off-policy reinforcement learning (RL) methods from Chapter 2 provide an effective way to learn optimal policies from static data: by learning value-functions, Q-learning and actor-critic algorithms, can learn optimal policies from even sub-optimal offline data. By attempting to isolate practical problems that hinder the usability of off-policy RL methods in learning from entirely static data, we wish to highlight challenges in offline RL. Specifically, we focus on answering the following questions:

(1) What is the effect of distributional shift?

The standard formulation of Q-learning and actor-critic prescribes a learning procedure that must make accurate counterfactul predictions about on states and actions visited by the learned policy. In general, the distribution of the learned policy will always be very different from that of the behavioral policy, and the difference will only be exacerbated for a correctly functioning algorithm that is able to find a policy close to the optimal

policy for the problem. We perform controlled experiments to investigate the amount of distributional shift and its impact on performance, observing that deviating away from the distributions of states and actions in the offline dataset can lead to significant instability over the course of learning.

(2) What is the effect of sampling error?

In general, it is impossible to precisely infer the true underlying dynamical system using just a finite amount of offline data. This means that akin to standard supervised learning, off-policy RL algorithms that reuse a static dataset for learning would also overfit to the training data. To what extent, does this overfitting hurt performance? We experimentally show that overfitting exists in practice by performing ablation studies on the number of gradient steps utilized for learning, and by demonstrating that oracle based early stopping techniques can be used to improve performance of Q-learning algorithms.

3.2 EXPERIMENTAL SETUP

While it is definitely possible to study challenges with off-policy RL methods in offline RL on common deep RL benchmark tasks (e.g., OpenAI Gym [262] environments), these tasks do not necessarily provide us with the ability to compute oracle solutions and isolate challenges individually. Therefore, we perform our study in a "unit-testing" setup, consisting of gridworld and other tabular environments with varying difficulty levels, where it is possible to compute oracle solutions and control for different factors independently.

For our study, we selected eight tabular domains, each with different qualitative attributes, including: gridworlds of varying sizes and observations, blind Cliffwalk [286], discretized Pendulum and Mountain Car based on OpenAI Gym [262], and a sparsely connected graph. We discuss each domain in detail below:

- **Gridworlds**. The Gridworld environment is an NxN grid with randomly placed walls. The reward is proportional to Manhattan distance to a goal state (1 at the goal, 0 at the initial position), and there is a 5% chance the agent travels in a different direction than commanded. We vary two parameters: the size (16×16 and 64×64), and the state representations. We use a one-hot representation, an (X, Y) coordinate tuple (represented as two one-hot vectors), and a random representation, a vector drawn from $\mathcal{N}(0,1)^N$, where N is the width or height of the Gridworld. The random observation significantly increases the challenge of function approximation, as significant state aliasing occurs.
- **Cliffwalk**: Cliffwalk is a toy example from Schaul et al. [286]. It consists of a sequence of states, where each state has two allowed actions: advance to the next

state or return to the initial state. A reward of 1.0 is obtained when the agent reaches the final state. Observations consist of vectors drawn from $\mathcal{N}(0, 1)^{16}$.

- **InvertedPendulum and MountainCar**: InvertedPendulum and MountainCar are discretized versions of continuous control tasks found in OpenAI gym [262], and are based on problems from classical RL literature. In the InvertedPendulum task, an agent must swing up an pendulum and hold it in its upright position. The state consists of the angle and angular velocity of the pendulum. Maximum reward is given when the pendulum is upright. The observation consists of the sin and cos of the pendulum angle, and the angular velocity. In the MountainCar task, the agent must push a vehicle up a hill, but the hill is steep enough that the agent must gather momentum by swinging back and forth within a valley in order to reach the top. The state consists of the position and velocity of the vehicle.
- **SparseGraph**: The SparseGraph environment is a 256-state graph with randomly drawn edges. Each state has two edges, each corresponding to an action. One state is chosen as the goal state, where the agent receives a reward of one.

In order to provide consistent metrics across domains, we normalize returns and errors involving Q-functions by the returns of the expert policy π^* on each environment.

3.3 Impact of distributional shift

Off-policy RL methods applied to the offline RL problem would typically attempt to learn an optimal policy, even though the static dataset may not be generated from an optimal policy. As a result, one issue that emerges is that of *distributional shift*: while these methods train a model of the value-function and the policy only using state-action tuples from the data, upon deployment, these models will need to make correct predictions on states and actions sampled from a different distribution of the learned policy. In general, models trained via machine learning are not robust when the distribution of inputs changes, indicating that distributional shift can be a challenge for off-policy RL methods. Is distributional shift a challenge in offline RL?

Indeed, theoretically, the effects of distributional shift have been quantified using the notion of a concentrability coefficient [235], a constant typically $\gg 1$, which provides a worst-case error bound on the performance of an off-policy RL method due to distributional shift. To evaluate if this challenge persists empirically as well, we will analyze Q-learning methods for various choices of data distributions in this section.

A crucial design decision we must make is to consider setups that do not confound distributional shift with access to limited data. Therefore, for our study, we provide the underlying algorithm oracle access to all state-action transitions in the MDP, but vary the *distribution* over state-action pairs from which these transitions are sampled.

3.3.1 *What Are the Best Data Distributions Without Sampling Error?*

We begin by studying the effect of data distributions when disentangled from sampling error. We run Q-learning with an ablation over various Q-function network architectures and data distributions and report our aggregate results in Fig. 2. Unif$(\boldsymbol{s}, \boldsymbol{a})$, Replay$(\boldsymbol{s}, \boldsymbol{a})$ (using a replay buffer consisting of data from a mixture of policies with different degrees of optimality), and Prioritized$(\boldsymbol{s}, \boldsymbol{a})$ (weighting induced by prioritized experience replay [286]) consistently result in the highest returns across all architectures. On the other hand, relatively "narrow" data distributions, such as those induced the optimal policy (π^*) or only using a mixture of a few policies (Replay(10)) results in poor performance. We believe that these results favor the *uniformity* hypothesis: intuitively, the best distributions spread weight across a larger support of the state-action space, reducing the amount of possible distributional shift. On the other hand, less-uniform distributions, such as the state-action distribution induced by the optimal policy, present multiple avenues to deviate away from the offline data distribution, resulting in larger distributional shift.

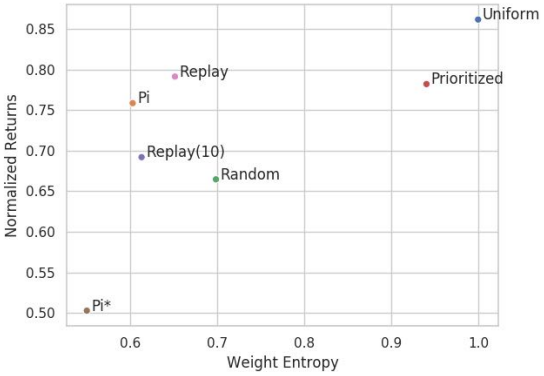


Figure 2: Normalized returns plotted against normalized entropy for different weighting distributions. All experiments assume access to all state-action pairs with a 256x256 Q-network. We see a general trend that high-entropy distributions lead to greater performance, corroborating the uniformity hypothesis.

To summarize, this indicates that narrow data distributions lead to worse performance compared to higher-entropy data distributions, indicating that distributional shift can have a significant impact on the performance of off-policy RL in the offline setting.

3.4 Impact of Sampling Error and Overfitting

So far, we have observed that the performance of off-policy RL algorithms based on Q-learning can be quite sensitive to the distribution of the offline data, even when all the transition tuples in the MDP are provided to the algorithm, and only the weighting over these samples is varied. In this section, we aim to study a distinct question: we investigate the performance of off-policy Q-learning when the offline dataset is of a finite size and may not contain all transitions in the MDP. To address any confounders from distributional shift, we

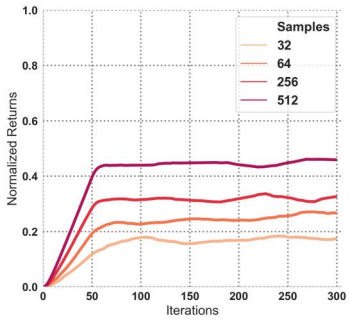
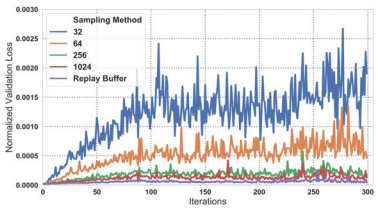
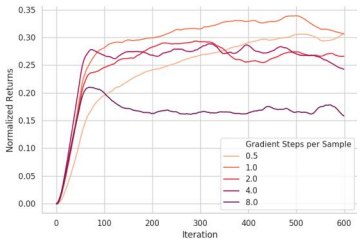


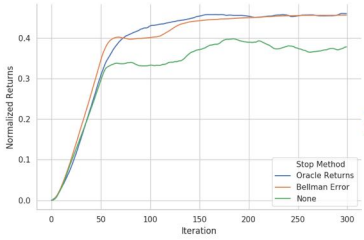
Figure 3: Samples vs. returns. More samples yields better performance.



(a) On-policy validation losses for varying amounts of on-policy data (or replay buffer), averaged across environments and seeds. Note that sampling from the replay buffer has lower on-policy validation loss, despite bias from distribution shift.



(b) Normalized returns plotted over training iterations (32 samples per iteration), for different ratios of gradient steps per sample. We observe that intermediate values of gradient steps work best, and too many gradient steps hurt performance.



(c) Normalized returns plotted over training iterations (32 samples are taken per iteration), for different early stopping methods. We observe that using proper early stopping can result in a modest performance increase.

provide the underlying Q-learning algorithm oracle access to collecting data from the current snapshot of the learned policy. While this sort of active data collection is not possible in the offline RL problem setting, this simplification allows us to more carefully localize the challenges of overfitting and sampling error.

We analyze several key points that relate to sampling error. First, we show that Q-learning is prone to overfitting, and this overfitting has a real impact on performance.

3.4.1 *Quantifying Overfitting*

We first quantify the amount of overfitting during training, by varying the number of samples. In order provide comparable validation errors across different experiments, we fix a reference sequence of Q-functions, $Q^1, ..., Q^N$, obtained during an arbitrary run of Q-learning. We then retrace the training sequence, and minimize the projection error $\Pi_\mu(Q^t)$ at each training iteration, using varying amounts of on-policy data or sampling from a replay buffer. We measure the validation error (the expected Bellman error) at each iteration under the on-policy distribution, plotted in Figure 4a. We note the obvious trend that more samples leads to significantly lower validation loss.

Next, Figure 3 shows the relationship between number of samples and returns. We see that more samples leads to improved learning speed and a better final solution. Despite overfitting being an issue, larger architectures still perform better because the bias introduced by smaller architectures dominates. This observation indicate that the nature of overfitting in RL is likely significantly different from that of supervised learning: while overfitting in supervised learning can bne controlled by regulating model capacity, off-policy RL methods likely need to rely on alternative techniques to control overfitting. We made some progress towards understanding this questions in Li et al. [209].

3.4.2 *Does Compensating for Overfitting Improve Performance?*

Finally, to confirm our hypotheses regarding overfitting, we now wish to understand if compensating for overfitting does lead to improved performance. One common technique for reducing overfitting is to utilize *early stopping* methods to mitigate overfitting without reducing model size. In order to understand whether early stopping criteria can reduce overfitting, we employ *oracle* stopping rules to provide an "upper bound" on the best potential improvement. We try two criteria for setting the number of gradient steps: the expected Bellman error and the expected returns of the greedy policy (oracle returns). We implement both methods by minimizing the TD-error to convergence, and retroactively selecting the intermediate Q-function which is judged best by the evaluation metric. Using oracle stopping metrics results in a modest boost in performance in tabular domains (Fig. 4c). Thus, we believe that there is promise in further improving such early-stopping methods for reducing overfitting in deep RL algorithms.

To summarize, we can draw a few conclusions from our experiments in this section. First, overfitting is indeed an issue with Q-learning, and too many gradient steps or too few samples can lead to poor performance. Second, although overfitting is a problem, large architectures are still preferred, because the bias from function approximation outweighs the increased overfitting from using large models.

Finally, we also remark that attempting to learn optimal policies from a static offline dataset will present both distributional shift and overfitting challenges together, and we would expect these to be tightly coupled with each other in the following sense: the inability to correclty identify the underlying MDP from finite samples in the offline dataset will induce errors in policy optimization, and these errors would then be exacerbated in the next step of learning as the learned policy deviates from the training data. This will also be evident in the theoretical results that we will present in subsequent chapters.

ACKNOWLEDGEMENTS AND FUNDING

We thank Vitchyr Pong and Kristian Hartikainen for providing us with implementations of RL algorithms. We thank Csaba Szepesvári and Chelsea Finn for comments on an earlier draft of this paper. SL thanks George Tucker for helpful discussion. We thank Google, NVIDIA, and Amazon for providing computational resources. This research was supported by Berkeley DeepDrive, NSF IIS-1651843 and IIS-1614653, the DARPA Assured Autonomy program, and ARL DCIST CRA W911NF-17-2-0181.

Part II

ALGORITHMIC APPROACHES TO OFFLINE RL

4

RESTRICTING ACTION SELECTION FOR POLICY LEARNING

Abstract

As we observed in Chapter 3, commonly used off-policy RL based on Q-learning can be highly sensitive to the data distribution, and are able to only make limited progress without collecting additional on-policy data. As a first step towards more robust off-policy algorithms that can be effective for offline RL, in this chapter, we identify *bootstrapping error* as a key source of instability in current methods. Bootstrapping error arises due to bootstrapping from actions that lie outside of the training data distribution, and it accumulates via the Bellman backup operator. We theoretically analyze bootstrapping error, and demonstrate how carefully constraining action selection in the Bellman backup can mitigate it. Based on our analysis, we propose a practical algorithm, bootstrapping error accumulation reduction (BEAR). We demonstrate that BEAR is able to learn robustly from different data distributions, including random and suboptimal offline data, on a range of control tasks.

4.1 INTRODUCTION

While state-of-the-art off-policy RL methods (e.g., [125, 236, 159, 75]) have demonstrated sample-efficient performance on complex tasks in robotics [159] and in simulation [325], we show that these methods fail completely to learn when presented with arbitrary offline datasets, corroborating our findings on simpler diagnostic tasks from Chapter 3. Since this issue persists even when the off-policy data comes from effective expert policies, which in principle should address any exploration challenge [55, 95, 90]. In this chapter, we aim to understand the root cause behind this inability and then develop an off-policy RL method that can learn effectively from static, offline datasets.

We find that a crucial challenge in applying value-based methods to off-policy scenarios arises in the bootstrapping process employed, when Q-functions are evaluated on out of **out-of-distribution** action inputs for computing the backup when training from off-

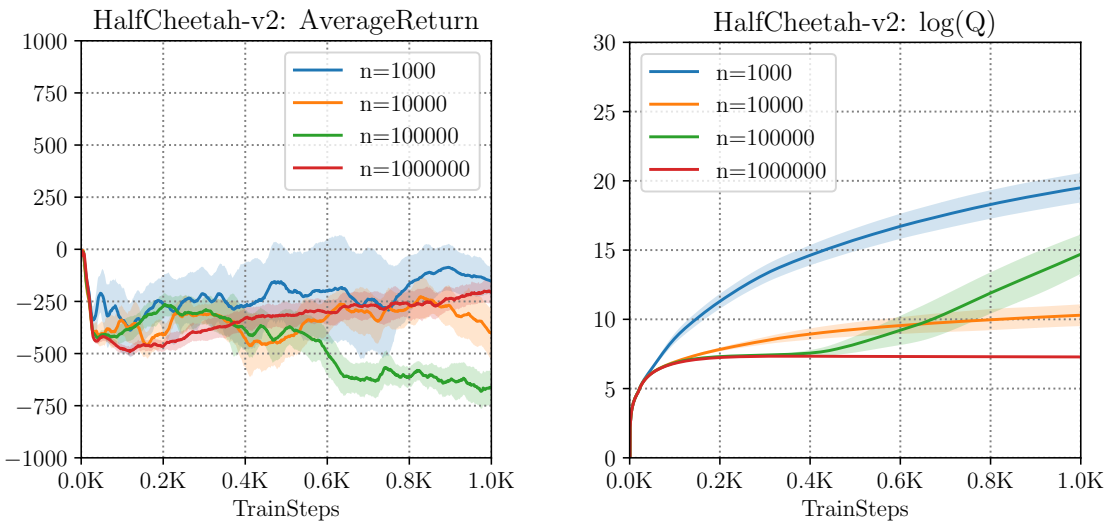


Figure 5: Performance of SAC on HalfCheetah-v2 (return (left) and log Q-values (right)) with off-policy expert data w.r.t. number of training samples (n). Note the large discrepancy between returns (which are negative) and log Q-values (which have large positive values), which is not solved with additional samples.

policy data. This may introduce errors in the Q-function and the algorithm is unable to collect new data in order to remedy those errors, making training unstable and divergent. We will first formalize and analyze the reasons for instability and poor performance when learning from off-policy data. Then, we will show that, through careful action selection, error propagation through the Q-function can be mitigated. We will then propose a principled algorithm called *bootstrapping error accumulation reduction* (BEAR) to control bootstrapping error in practice, which uses the notion of *support-set matching* to prevent error accumulation. Finally, through systematic experiments, we will show the effectiveness of our method on continuous-control MuJoCo Gym tasks, with a variety of off-policy datasets: generated by a random, suboptimal, or optimal policies. BEAR is consistently robust to the training dataset, matching or exceeding the state-of-the-art in all cases, whereas existing algorithms only perform well for specific datasets.

4.2 Out-of-Distribution Actions in Q-Learning

Building upon the discoveries presented in Chapter 3, we have observed that Q-learning methods frequently struggle to learn from static, off-policy data, particularly in high-dimensional tasks. This difficulty is evident in Figure 5. As outlined in the discussion of Chapter 3, the primary cause of this instability lies in the challenges of handling distributional shift and statistical error. However, a crucial question remains: which factor plays a more significant role in this example? Furthermore, what mechanism underlies the emergence of this instability?

First of all, note that increasing the size of the static dataset does not rectify the problem (compare $n = 1M$ vs $n = 1000$), suggesting the issue in this setting is largely due

to distributional shift. We can understand the source of this instability by examining the form of the Bellman backup. Although minimizing the mean squared Bellman error corresponds to a supervised regression problem, the targets for this regression are themselves derived from the current Q-function estimate. The targets are calculated by maximizing the learned Q-values with respect to the action at the next state. However, the Q-function estimator is only reliable on inputs from the same distribution as its training set. As a result, naïvely maximizing the value may evaluate the $\hat{Q}$ estimator on actions that lie far outside of the training distribution, resulting in pathological values that incur large error. We refer to these actions as out-of-distribution (OOD) actions.

Formally, let $\zeta_k(\boldsymbol{s}, \boldsymbol{a}) = |Q_k(\boldsymbol{s}, \boldsymbol{a}) - Q^*(\boldsymbol{s}, \boldsymbol{a})|$ denote the total error at iteration k of Q-learning, and let $\delta_k(\boldsymbol{s}, \boldsymbol{a}) = |Q_k(\boldsymbol{s}, \boldsymbol{a}) - \mathcal{B}Q_{k-1}(\boldsymbol{s}, \boldsymbol{a})|$ denote the current Bellman error. Then, we have $\zeta_k(\boldsymbol{s}, \boldsymbol{a}) \leq \delta_k(\boldsymbol{s}, \boldsymbol{a}) + \gamma \max_{\boldsymbol{a}'} \mathbb{E}_{\boldsymbol{s}'}[\zeta_{k-1}(\boldsymbol{s}', \boldsymbol{a}')]$. In other words, errors from $(\boldsymbol{s}', \boldsymbol{a}')$ are discounted, then accumulated with new errors $\delta_k(\boldsymbol{s}, \boldsymbol{a})$ from the current iteration. We expect $\delta_k(\boldsymbol{s}, \boldsymbol{a})$ to be high on OOD states and actions, as errors at these state-actions are never directly minimized while training.

To mitigate bootstrapping error, we can restrict the policy to ensure that it output actions that lie in the support of the training distribution. This is distinct from previous work [149] which implicitly constrains the *distribution* of the learned policy to be close to the behavior policy, similarly to behavioral cloning [284]. While this is sufficient to ensure that actions lie in the training set with high probability, it is overly restrictive. For example, if the behavior policy is close to uniform, the learned policy will behave randomly, resulting in poor performance, even when the data is sufficient to learn a strong policy (see Figure 6 for an illustration). Formally, this means that a learned policy $\pi(\boldsymbol{a}|\boldsymbol{s})$ has positive density *only where* the density of the behaviour policy $\beta(\boldsymbol{a}|s)$ is more than a threshold (i.e., $\forall \boldsymbol{a}, \beta(\boldsymbol{a}|\boldsymbol{s}) \leq \varepsilon \implies \pi(\boldsymbol{a}|\boldsymbol{s}) = 0$), instead of a closeness constraint on the value of the density $\pi(\boldsymbol{a}|\boldsymbol{s})$ and $\beta(\boldsymbol{a}|\boldsymbol{s})$. Our analysis instead reveals a tradeoff between staying within the data distribution and finding a suboptimal solution when the constraint is too restrictive. Our analysis motivates us to restrict the support of the learned policy, but not the probabilities of the actions lying within the support. This avoids evaluating the Q-function estimator on OOD actions, but remains flexible in order to find a performant policy. Our proposed algorithm leverages this insight.

4.3 Formal Analysis and Distribution-Constrained Backups

In this section, we define and analyze a backup operator that restricts the set of policies used in the maximization of the Q-function, and we derive performance bounds which depend on the restricted set. This provides motivation for constraining policy support to the data distribution, allowing us to address the issue discussed above. We begin with the definition of a distribution-constrained operator:

Definition 4.3.1 (Distribution-constrained operators)**.** *Given a set of policies* Π*, the distribution-constrained backup operator is defined as:*

$$\mathcal{B}^{\Pi}Q(\boldsymbol{s},\boldsymbol{a}) \stackrel{\text{def}}{=} \mathbb{E}\left[R(\boldsymbol{s},\boldsymbol{a}) + \gamma \max_{\pi\in\Pi} \mathbb{E}_{T(\boldsymbol{s}'|\boldsymbol{s},\boldsymbol{a})}\left[V(\boldsymbol{s}')\right]\right] \qquad V(\boldsymbol{s}) \stackrel{\text{def}}{=} \max_{\pi\in\Pi} \mathbb{E}_{\pi}[Q(\boldsymbol{s},\boldsymbol{a})] \ .$$

This backup operator satisfies properties of the standard Bellman backup, such as convergence to a fixed point, as discussed in Appendix B.1. To analyze the (sub)optimality of performing this backup under approximation error, we first quantify two sources of error. The first is a *suboptimality bias*. The optimal policy may lie outside the policy constraint set, and thus a suboptimal solution will be found. The second arises from distribution shift between the training distribution and the policies used for backups. This formalizes the notion of OOD actions. To capture suboptimality in the final solution, we define a *suboptimality constant*, which measures how far π^* is from Π.

Definition 4.3.2 (Suboptimality constant)**.** *The suboptimality constant is defined as:*

$$\alpha(\Pi) = \max_{\boldsymbol{s},\boldsymbol{a}} |\mathcal{B}^{\Pi}Q^*(\boldsymbol{s},\boldsymbol{a}) - \mathcal{B}Q^*(\boldsymbol{s},\boldsymbol{a})|.$$

Next, we define a concentrability coefficient [235], which quantifies how far the visitation distribution generated by policies from Π is from the training data distribution. This constant captures the degree to which states and actions are out of distribution.

Assumption 4.3.3 (Concentrability)**.** *Let* ρ_0 *denote the initial state distribution, and* $\mu(\boldsymbol{s},\boldsymbol{a})$ *denote the distribution of the training data over* $\mathcal{S}\times\mathcal{A}$*, with marginal* $\mu(\boldsymbol{s})$ *over* $\mathcal{S}$*. Suppose there exist coefficients* $c(k)$ *such that for any* $\pi_1, ...\pi_k \in \Pi$ *and* $s \in \mathcal{S}$*:*

$$\rho_0 P^{\pi_1} P^{\pi_2} ... P^{\pi_k}(s) \leq c(k)\mu(\boldsymbol{s}),$$

where P^{π_i} *is the transition operator on states induced by* π_i*. Then, define the concentrability coefficient* $C(\Pi)$ *as*

$$C(\Pi) \stackrel{\text{def}}{=} (1-\gamma)^2 \sum_{k=1}^{\infty} k\gamma^{k-1}c(k).$$

To provide some intuition for $C(\Pi)$, if μ was generated by a single policy π, and $\Pi = \{\pi\}$ was a singleton set, then we would have $C(\Pi) = 1$, which is the smallest possible value. However, if Π contained policies far from π, the value could be large, potentially infinite if the support of Π is not contained in π. Now, we bound the performance of approximate distribution-constrained Q-iteration:

Theorem 4.3.4. *Suppose we run approximate distribution-constrained value iteration with a set constrained backup $\mathcal{B}^{\Pi}$. Assume that $\delta(\boldsymbol{s}, \boldsymbol{a}) \geq \max_k |Q_k(\boldsymbol{s}, \boldsymbol{a}) - \mathcal{B}^{\Pi} Q_{k-1}(\boldsymbol{s}, \boldsymbol{a})|$ bounds the Bellman error. Then,*

$$\lim_{k \to \infty} \mathbb{E}_{\rho_0}[|V^{\pi_k}(\boldsymbol{s}) - V^*(\boldsymbol{s})|] \leq \frac{\gamma}{(1-\gamma)^2} \left[C(\Pi) \mathbb{E}_{\mu}[\max_{\pi \in \Pi} \mathbb{E}_{\pi}[\delta(\boldsymbol{s}, \boldsymbol{a})]] + \frac{1-\gamma}{\gamma} \alpha(\Pi) \right]$$

Proof. See Appendix B.2, Theorem B.2.1 □

This bound formalizes the tradeoff between keeping policies chosen during backups close to the data (captured by $C(\Pi)$) and keeping the set Π large enough to capture well-performing policies (captured by $\alpha(\Pi)$). When we expand the set of policies Π, we are increasing $C(\Pi)$ but decreasing $\alpha(\Pi)$. An example of this tradeoff, and how a careful choice of Π can yield superior results, is given in a tabular gridworld example in Fig. 6, where we visualize errors accumulated during distribution-constrained Q-iteration for different choices of Π.

Finally, we motivate the use of support sets to construct Π. We are interested in the case where $\Pi_\epsilon = \{\pi \mid \pi(\boldsymbol{a}|\boldsymbol{s}) = 0 \text{ whenever } \beta(\boldsymbol{a}|\boldsymbol{s}) < \epsilon\}$, where β is the behavior policy (i.e., Π is the set of policies that have support in the probable regions of the behavior policy). Defining Π_ϵ in this way allows us to bound the concentrability coefficient:

Theorem 4.3.5. *Assume the data distribution μ is generated by a behavior policy β. Let $\mu(\boldsymbol{s})$ be the marginal state distribution under the data distribution. Define $\Pi_\epsilon = \{\pi \mid \pi(\boldsymbol{a}|\boldsymbol{s}) = 0$ whenever $\beta(\boldsymbol{a}|\boldsymbol{s}) < \epsilon\}$ and let μ_{Π_ϵ} be the highest discounted marginal state distribution starting from the initial state distribution ρ and following policies $\pi \in \Pi_\epsilon$ at each time step thereafter. Then, there exists a concentrability coefficient $C(\Pi_\epsilon)$ which is bounded:*

$$C(\Pi_\epsilon) \leq C(\beta) \cdot \left(1 + \frac{\gamma}{(1-\gamma) f(\epsilon)} (1-\epsilon)\right)$$

where $f(\epsilon) \stackrel{\text{def}}{=} \min_{\boldsymbol{s} \in \mathcal{S}, \mu_{\Pi_\epsilon}(\boldsymbol{s}) > 0} [\mu(\boldsymbol{s})] > 0$.

Proof. See Appendix B.2, Theorem B.2.2 □

Qualitatively, $f(\epsilon)$ is the minimum discounted visitation marginal of a state under the behaviour policy if only actions which are more than ϵ likely are executed in the environment. Thus, using support sets gives us a single lever, ϵ, which simultaneously trades off the value of $C(\Pi)$ and $\alpha(\Pi)$. Not only can we provide theoretical guarantees, we will see in our experiments (Sec. 4.5) that constructing Π in this way provides a simple and effective method for implementing distribution-constrained algorithms.

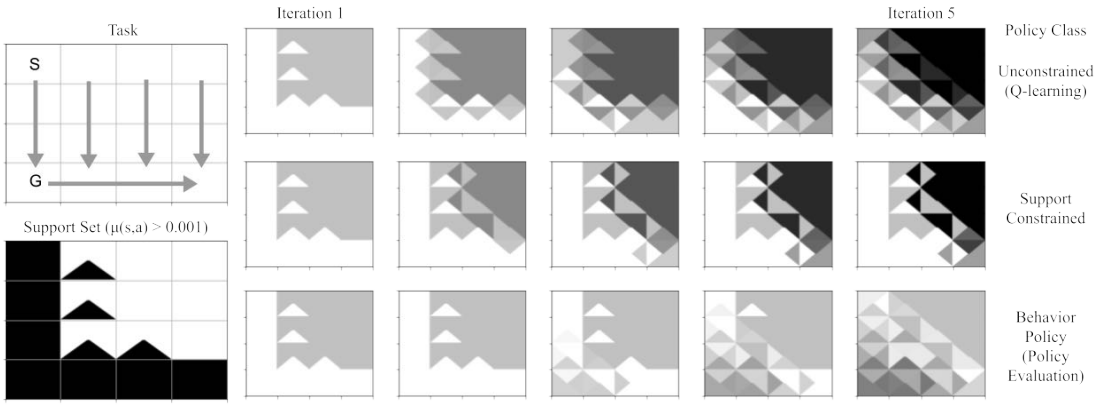


Figure 6: Visualized error propagation in Q-learning for various choices of the constraint set Π: unconstrained (top row) distribution-constrained (middle), and constrained to the behavior policy (policy-evaluation, bottom). Triangles represent Q-values for actions that move in different directions. The task (left) is to reach the bottom-left corner (G) from the top-left (S), but the behaviour policy (visualized as arrows in the task image, support state-action pairs are shown in black on the support set image) travels to the bottom-right with a small amount of ϵ-greedy exploration. Dark values indicate high error, and light values indicate low error. Standard backups propagate large errors from the low-support regions into the high-support regions, leading to high error. Policy evaluation reduces error propagation from low-support regions, but introduces significant suboptimality bias, as the data policy is not optimal. A carefully chosen distribution-constrained backup strikes a balance between these two extremes, by confining error propagation in the low-support region while introducing minimal suboptimality bias.

Intuitively, this means we can prevent an increase in overall error in the Q-estimate by selecting policies supported on the support of the training action distribution, which would ensure roughly bounded projection error $\delta_k(\boldsymbol{s}, \boldsymbol{a})$ while reducing the suboptimality bias, potentially by a large amount. Bounded error $\delta_k(\boldsymbol{s}, \boldsymbol{a})$ on the support set of the training distribution is a reasonable assumption when using highly expressive function approximators, such as deep networks, especially if we are willing to reweight the transition set [286, 90]. We further elaborate on this point in Appendix B.3.

4.4 Bootstrapping Error Accumulation Reduction (BEAR)

We now propose a practical actor-critic algorithm (built on the framework of TD3 [96] or SAC [125]) that uses distribution-constrained backups to reduce accumulation of bootstrapping error. The key insight is that we can search for a policy with the same support as the training distribution, while preventing accidental error accumulation. Our algorithm has two main components. Analogous to BCQ [96], we use K Q-functions and use the minimum Q-value for policy improvement, and design a constraint which will be used for searching over the set of policies Π_ϵ, which share the same support as the behavior policy. Both of these components will appear as modifications of the policy improvement step in actor-critic style algorithms. We also note that policy improvement

can be performed with the mean of the K Q-functions, and we found that this scheme works as good in our experiments.

We denote the set of Q-functions as: $\hat{Q}_1, \cdots, \hat{Q}_K$. Then, the policy is updated to maximize the conservative estimate of the Q-values within Π_ϵ:

$$\pi_\phi(\boldsymbol{s}) := \max_{\pi \in \Pi_\epsilon} \mathbb{E}_{a \sim \pi(\cdot|\boldsymbol{s})} \left[\min_{j=1,..,K} \hat{Q}_j(\boldsymbol{s}, \boldsymbol{a}) \right]$$

In practice, the behavior policy β is unknown, so we need an approximate way to constrain π to Π. We define a differentiable constraint that approximately constrains π to Π, and then approximately solve the constrained optimization problem via dual gradient descent. We use the sampled version of maximum mean discrepancy (MMD) [114] between the unknown behavior policy β and the actor π because it can be estimated based solely on samples from the distributions. Given samples $x_1, \cdots, x_n \sim P$ and $y_1, \cdots, y_m \sim Q$, the sampled MMD between P and Q is given by:

$$\text{MMD}^2(\{x_1, \cdots, x_n\}, \{y_1, \cdots, y_m\}) = \frac{1}{n^2} \sum_{i,i'} k(x_i, x_{i'}) - \frac{2}{nm} \sum_{i,j} k(x_i, y_j) + \frac{1}{m^2} \sum_{j,j'} k(y_j, y_{j'}).$$

Here, $k(\cdot, \cdot)$ is any universal kernel. In our experiments, we find both Laplacian and Gaussian kernels work well. The expression for MMD does not involve the density of either distribution and it can be optimized directly through samples. Empirically we find that, in the low-intermediate sample regime, the sampled MMD between P and Q is similar to the MMD between a uniform distribution over P's support and Q, which makes MMD roughly suited for constraining distributions to a given support set. (See Appendix B.3.3 for numerical simulations justifying this approach).

Putting together, the optimization problem in the policy improvement step is

$$\pi_\phi := \max_{\pi \in \Delta_{|S|}} \mathbb{E}_{\boldsymbol{s} \sim \mathcal{D}} \mathbb{E}_{\boldsymbol{a} \sim \pi(\cdot|\boldsymbol{s})} \left[\min_{j=1,..,K} \hat{Q}_j(\boldsymbol{s}, \boldsymbol{a}) \right] \text{ s.t. } \mathbb{E}_{\boldsymbol{s} \sim \mathcal{D}}[\text{MMD}(\mathcal{D}(\boldsymbol{s}), \pi(\cdot|\boldsymbol{s}))] \leq \varepsilon \tag{4.4.1}$$

where ε is an approximately chosen threshold. We choose a threshold of $\varepsilon = 0.05$ in our experiments. The algorithm is summarized in Algorithm 3.

How does BEAR connect with distribution-constrained backups described in Section 4.1? Step 5 of the algorithm restricts π_ϕ to lie in the support of β. This insight is formally justified in Theorems 4.1 & 4.2 ($C(\Pi_\varepsilon)$ is bounded). Computing distribution-constrained backup exactly by maximizing over $\pi \in \Pi_\varepsilon$ is intractable in practice. As an approximation, we sample Dirac policies in the support of β (Alg 1, Line 5) and perform empirical maximization to compute the backup. As the maximization is performed over

a *narrower* set of Dirac policies ($\{\delta_{\boldsymbol{a}_i}\} \subseteq \Pi_\varepsilon$), the bound in the above Theorem still holds. Empirically, we show in Section 4.5 that this approximation is sufficient to outperform previous methods. This connection is briefly discussed in Appendix B.3.2.

Algorithm 3 Q-learning variant of BEAR (BEAR)

Dataset $\mathcal{D}$, target network rate τ, batch size N, sampled actions for MMD n, minimum λ
Initialize Q-ensemble $\{Q_{\theta_i}\}_{i=1}^K$, actor π_ϕ, multiplier α, target networks $\{Q_{\theta_i'}\}_{i=1}^K$, and a target actor $\pi_{\phi'}$, with $\phi' \leftarrow \phi, \theta_i' \leftarrow \theta_i$
for all t in $\{1, \ldots, \mathrm{N}\}$ **do**
    Sample mini-batch of transitions $(\boldsymbol{s}, \boldsymbol{a}, r, \boldsymbol{s}') \sim \mathcal{D}$
Q-update:
    Sample p action samples, $\{\boldsymbol{a}_i \sim \pi_{\phi'}(\cdot|\boldsymbol{s}')\}_{i=1}^p$
    Define $y(\boldsymbol{s}, \boldsymbol{a}) := \max_{\boldsymbol{a}_i}[\lambda \min_{j=1,..,K} Q_{\theta_j'}(\boldsymbol{s}', \boldsymbol{a}_i) + (1-\lambda) \max_{j=1,..,K} Q_{\theta_j'}(\boldsymbol{s}', \boldsymbol{a}_i)]$
    $\forall i, \theta_i \leftarrow \arg\min_{\theta_i}(Q_{\theta_i}(\boldsymbol{s}, \boldsymbol{a}) - (r + \gamma y(\boldsymbol{s}, \boldsymbol{a})))^2$
Policy-update:
    Sample actions $\{\hat{\boldsymbol{a}}_i \sim \pi_\phi(\cdot|\boldsymbol{s})\}_{i=1}^m$ and $\{\boldsymbol{a}_j \sim \mathcal{D}(\boldsymbol{s})\}_{j=1}^n$.
    Update ϕ, α by minimizing Equation 4.4.1 with Lagrange multiplier α.
    Update Target Networks: $\theta_i' \leftarrow \tau\theta_i + (1-\tau)\theta_i'$; $\phi' \leftarrow \tau\phi + (1-\tau)\phi'$
end for

In summary, the actor is updated towards maximizing the Q-function while still being constrained to remain in the valid search space defined by Π_ϵ. The Q-function uses actions sampled from the actor to then perform distribution-constrained Q-learning, over a reduced set of policies. At test time, we sample p actions from $\pi_\phi(\boldsymbol{s})$ and the Q-value maximizing action out of these is executed in the environment. Implementation and other details are present in Appendix B.4.

4.5 Experimental evaluation of BEAR

In our experiments, we study how BEAR performs when learning from static offline data on a variety of continuous control benchmark tasks. We evaluate our algorithm in three settings: when the dataset $\mathcal{D}$ is generated by **(1)** a completely random behavior policy, **(2)** a partially trained, medium scoring policy, and **(3)** an optimal policy. Condition **(2)** is of particular interest, as it captures many common use-cases in practice, such as learning from imperfect demonstration data (e.g., of the sort that are commonly available for autonomous driving [99]), or reusing previously collected experience during off-policy RL. We compare our method to several prior methods: a baseline actor-critic algorithm (TD3), the BCQ algorithm [95], which aims to address a similar problem, as discussed in Section 4.2, KL-control [150] (which solves a KL-penalized RL problem similarly to maximum entropy RL), a static version of DQfD [138], and a behavior cloning (BC) baseline, which simply imitates the data distribution. This serves to measure whether

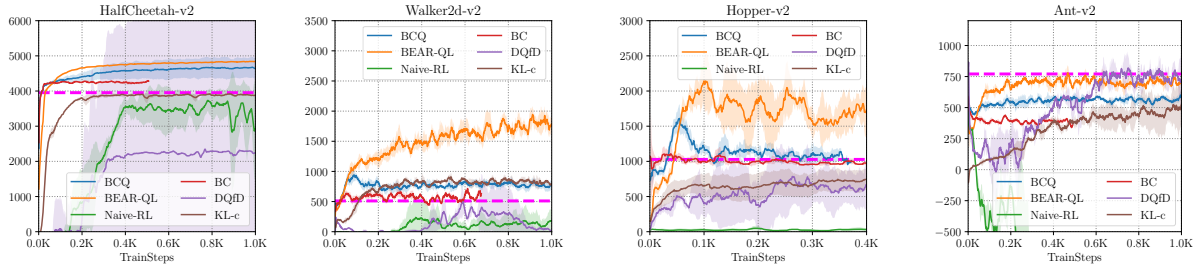


Figure 7: Average performance of BEAR, BCQ, Naïve RL and BC on medium-quality data averaged over 5 seeds. BEAR outperforms both BCQ and Naïve RL. Average return over the training data is indicated by the magenta line. One step on the x-axis corresponds to 1000 gradient steps.

each method actually performs effective RL, or simply copies the data. We report the average evaluation return over 5 seeds of the policy given by the learned algorithm, in the form of a learning curve as a function of number of gradient steps taken by the algorithm. These samples are only collected for evaluation, and are not used for training.

4.5.1 *Performance on Medium-Quality Data*

We first discuss the evaluation of condition with "mediocre" data **(2)**, as this condition resembles the settings where we expect training on offline data to be most useful. We collected one million transitions from a partially trained policy, so as to simulate imperfect demonstration data or data from a mediocre prior policy. In this scenario, we found that BEAR consistently outperforms BCQ [95] and a naïve off-policy RL baseline (TD3) (by large margins), as shown in Figure 7. This scenario is the most relevant from an application point of view, as access to optimal data may not be feasible, and random data might have inadequate exploration to efficient learn a good policy. We also evaluate the accuracy with which the learned Q-functions predict actual policy returns. These trends are provided in Appendix B.5. Our KL-control baseline uses automatic temperature tuning [125]. We find that KL-control usually performs similar or worse to BC, whereas DQfD tends to diverge, and often exhibits a huge variance across different runs (for example, HalfCheetah-v2 environment).

4.5.2 *Performance on Random and Optimal Datasets*

In Figure 9, we show the performance of each method when trained on data from a random policy (top) and a near-optimal policy (bottom). In both cases, our method BEAR achieves good results, consistently exceeding the average dataset return on random data, and matching the optimal policy return on optimal data. Naïve RL also often does well on random data. For a random data policy, all actions are in-distribution, since they all have equal probability. This is consistent with our hypothesis that OOD actions are one of the

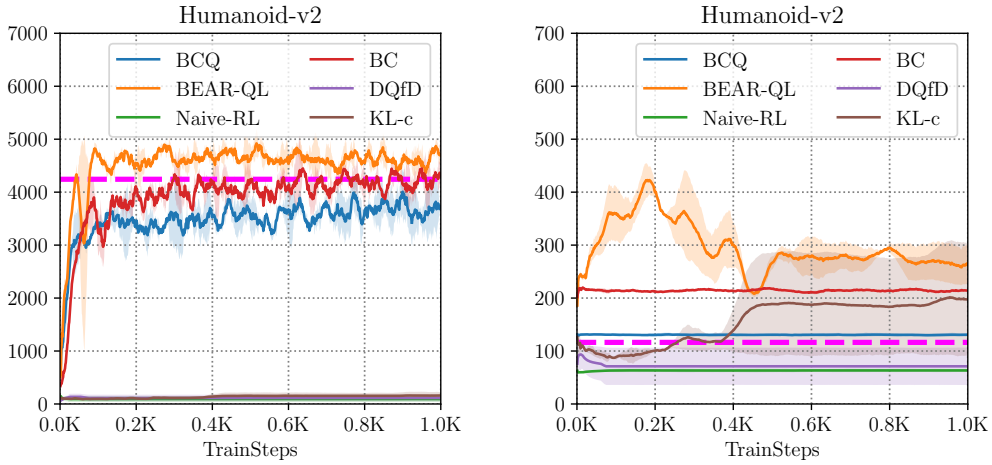


Figure 8: Performance of BEAR, BCQ, Naïve RL and BC on medium-quality (left) and random (right) data in the Humanoid-v2 environment. Note that BEAR outperforms prior methods.

main sources of error in off-policy learning on static datasets. The prior BCQ method [95] performs well on optimal data but performs poorly on random data, where the constraint is too strict. These results show that BEAR is more robust to the dataset composition, and can learn consistently in a variety of settings. We find that KL-control and DQfD can be unstable in these settings. Finally, in Figure 8, we show that BEAR outperforms other considered prior methods in the challenging Humanoid-v2 environment as well, in two cases – Medium-quality data and random data.

4.6 Related Work

Errors arising from inadequate sampling, distributional shift, and function approximation have been rigorously studied as "error propagation" in approximate dynamic programming (ADP) [27, 234, 79, 288]. These works often study how Bellman errors accumulate and propagate to nearby states via bootstrapping. In this chapter, we build upon tools from this analysis to show that performing Bellman backups on static datasets leads to error accumulation due to out-of-distribution values. Our approach is motivated as reducing the rate of propagation of error propagation between states.

BEAR constrains actor updates so that the actions remain in the support of the training dataset. Several works have explored similar ideas in the context of off-policy learning learning in online settings. Kakade and Langford [158] shows that large policy updates can be destructive, and propose a conservative policy iteration scheme which constrains actor updates to be small for provably convergent learning. Grau-Moya et al. [112] use a learned prior over actions in the maximum entropy RL framework [201] and justify it as a regularizer based on mutual information. However, none of these methods use static

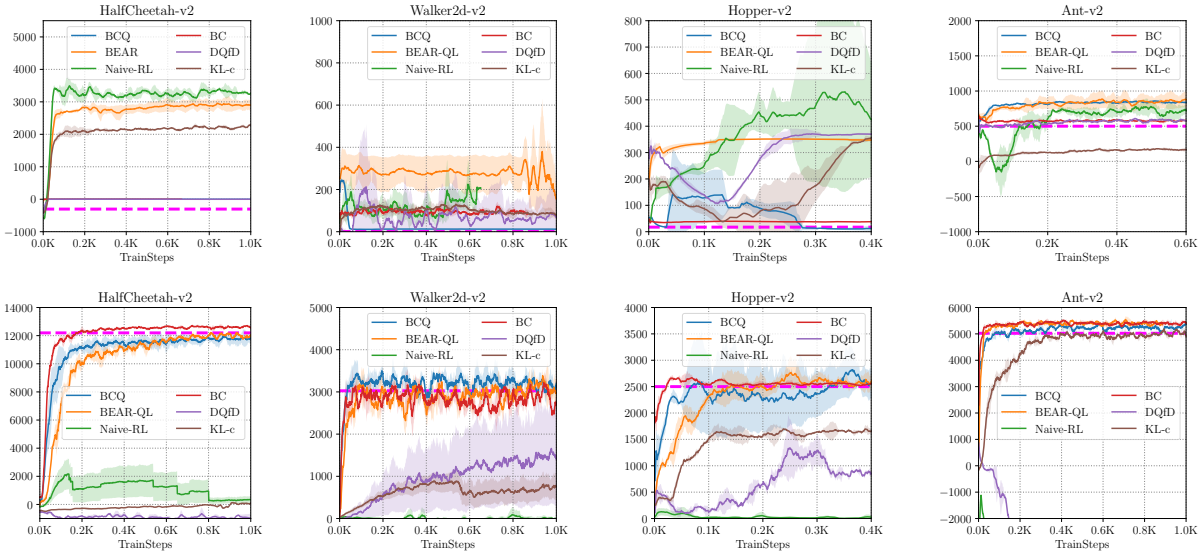


Figure 9: Average performance of BEAR, BCQ, Naïve RL and BC on random data (top row) and optimal data (bottom row) over 5 seeds. BEAR is the only algorithm capable of learning in both scenarios. Naïve RL cannot handle optimal data, since it does not illustrate mistakes, and BCQ favors a behavioral cloning strategy (performs quite close to behavior cloning in most cases), causing it to fail on random data. Average return over the training dataset is indicated by the dashed magenta line.

datasets. Importance Sampling based distribution re-weighting [236, 102, 270, 225] has also been explored primarily in the context of off-policy policy evaluation.

Most closely related to our approach is batch-constrained Q-learning (BCQ) [95] and SPIBB [193], which also discuss instability arising from previously unseen actions. Fujimoto et al. [95] show convergence properties of an action-constrained Bellman backup operator in tabular, error-free settings. We prove stronger results under approximation errors and provide a bound on the *suboptimality* of the solution. This is crucial as it drives the design choices for a practical algorithm. As a consequence, although we experimentally find that [95] outperforms standard Q-learning methods when the off-policy data is collected by an expert, BEAR outperforms [95] when the off-policy data is collected by a suboptimal policy, as is common in real-life applications. SPIBB [193], like BEAR, is an algorithm based on constraining the learned policy to the support of a behavior policy. However, the authors of SPIBB do not extend safe performance guarantees from the batch-constrained case to the relaxed support-constrained case, and do not study high-dimensional control tasks.

4.7 Discussion and Limitations

In this chapter, we theoretically and empirically analyzed how error propagates in offline RL due to the use of out-of-distribution actions for computing the target values

in the Bellman backup. Our experiments suggest that this source of error is one of the primary issues afflicting off-policy RL: increasing the number of samples does not appear to mitigate the degradation issue (Figure 5), and training with naïve RL on data from a random policy, where there are no out-of-distribution actions, shows much less degradation than training on data from more focused policies (Figure 9). Armed with this insight, we develop a method for mitigating the effect of out-of-distribution actions, which we call BEAR. BEAR constrains the backup to use actions that have non-negligible support under the data distribution, but without being overly conservative in constraining the learned policy. We observe experimentally that BEAR achieves good performance across a range of tasks, and across a range of dataset compositions, learning well on random, medium-quality, and expert data.

Limitations. A limitation of BEAR, and also other approaches that perform constrained-action selection, is that they can be overly conservative compared to methods that aim to constrain state-distributions directly and, even, methods that utilize conservative value estimation, especially with datasets collected from mixtures of sub-optimal policies. In the next chapter, we will present a principle for conservative value-estimation and develop model-based and model-free offline RL algorithms based on this principle, which we find leads to better performance across a broader range of tasks.

Theoretically, while our analysis demonstrates that support-matching based on distributional backups can perform better than policy-constrained backups, unfortunately, developing an efficient practical method for support matching in high-dimensional continuous action spaces is still an open question. Additionally, there is also room to formally characterize the gap between support-matching and policy-constrained backups. We make progress towards this open question in a follow-up manuscript [300].

ACKNOWLEDGEMENTS AND FUNDING

We thank Kristian Hartikainen for sharing implementations of RL algorithms and for help in debugging certain issues. We thank Matthew Soh for help in setting up environments. We thank Aurick Zhou, Chelsea Finn, Abhishek Gupta, Kelvin Xu and Rishabh Agarwal for informative discussions. We thank Ofir Nachum for comments on an earlier draft of this paper. We thank Google, NVIDIA, and Amazon for providing computational resources. This research was supported by Berkeley DeepDrive, JPMorgan Chase & Co., NSF IIS-1651843 and IIS-1614653, the DARPA Assured Autonomy program, and ARL DCIST CRA W911NF-17-2-0181.

5

CONSERVATIVE VALUE FUNCTION ESTIMATION

Abstract

In Chapters 3 and 4, we discussed how standard off-policy RL methods can fail in offline RL and how restricting action selection can provide for a first approach to tackle distributional shift in offline RL. While this approach does attain promising empirical performance, methods based on this principle tend to be restrictive, constraining the learned policy to the behavioral policy even when the Q-function on out-of-distribution actions are not erroneously large to inhibit effective policy learning. In this chapter, we present a paradigm called *conservative value function estimation*, which aims to address these limitations by learning a value-function such that the expected value of a policy under this value-function lower-bounds its true value. We discuss two concrete variants of this paradigm: a model-free variant, called conservative Q-learning (CQL) [181], and a model-based variant, called conservative offline model-based policy optimization (COMBO) [366]. Both of these variants can be implemented by augmenting the standard temporal-difference error objective with a Q-value regularizer, on top of model-based and model-free Q-learning and actor-critic algorithms. We will also theoretically show that this paradigm produces conservative value functions, that lower bound the ground-truth value of the current policy, and policies, that enjoy safe policy improvement.

5.1 MODEL-FREE CONSERVATIVE VALUE FUNCTION ESTIMATION

Directly utilizing existing value-based off-policy RL algorithms in an offline setting generally results in poor performance, due to issues with bootstrapping from out-of-distribution actions [179, 95] and overfitting [90, 179, 6]. This typically manifests as erroneously optimistic value function estimates (observe the erroneously high Q-values in Figure 5). If we can instead learn a *conservative* estimate of the value function, which provides a lower bound on the true values, this overestimation problem could be addressed. We propose a novel method for learning such conservative Q-functions via a simple

modification to standard value-based RL algorithms. The key idea behind our method is to minimize values under an appropriately chosen distribution over state-action tuples. Our theoretical analysis of conservative Q-learning shows that the expected value of this Q-function under the policy lower-bounds the true policy value. We also empirically validate this observation and demonstrate the robustness of our approach to Q-function estimation error. Our practical algorithm uses these conservative estimates for policy evaluation and offline RL. CQL can be implemented as a modification on top of a number of standard, online RL algorithms [122, 51], simply by adding the CQL regularization terms to the Q-function update. In our experiments, we demonstrate the efficacy of CQL for offline RL, in domains with complex dataset compositions, where prior methods are typically known to perform poorly [86] and domains with high-dimensional visual inputs [22, 6].

5.1.1 *The Conservative Value Estimation Paradigm*

In this section, we develop a conservative value-estimation paradigm, such that the expected value of a policy under the learned Q-function lower-bounds its true value. Lower-bounded Q-values prevent the over-estimation that is common in offline RL settings due to OOD actions and approximation error [179]. We start with developing this paradigm in the context of model-free policy evaluation, which can be used by itself as an off-policy evaluation procedure, or integrated into a complete model-free offline RL algorithm, that we call conservative Q-learning (CQL), as we will discuss in Section 5.1.3. Finally, we will extend this algorithm to the model-based setting in Section 5.2.

5.1.2 *Conservative Off-Policy Evaluation*

We aim to estimate the value $V^\pi(\boldsymbol{s})$ of a target policy π given access to a dataset, $\mathcal{D}$, generated by following a behavior policy $\pi_\beta(\boldsymbol{a}|\boldsymbol{s})$. Because we are interested in preventing overestimation of the policy value, we learn a *conservative*, lower-bound Q-function by additionally minimizing Q-values alongside a standard Bellman error objective. Our choice of penalty is to minimize the expected Q-value under a particular distribution of state-action pairs, $\mu(\boldsymbol{s}, \boldsymbol{a})$. Since standard Q-function training does not query the Q-function value at unobserved states, but queries the Q-function at unseen actions, we restrict μ to match the state-marginal in the dataset, such that $\mu(\boldsymbol{s}, \boldsymbol{a}) = d^{\pi_\beta}(\boldsymbol{s})\mu(\boldsymbol{a}|\boldsymbol{s})$. This gives rise to the iterative update for training the Q-function:

$$\hat{Q}^{k+1} \leftarrow \arg\min_{Q} \; \alpha \, \mathbb{E}_{\boldsymbol{s}\sim\mathcal{D},\boldsymbol{a}\sim\mu(\boldsymbol{a}|\boldsymbol{s})}\left[Q(\boldsymbol{s}, \boldsymbol{a})\right] + \frac{1}{2}\, \mathbb{E}_{\boldsymbol{s},\boldsymbol{a}\sim\mathcal{D}}\left[\left(Q(\boldsymbol{s}, \boldsymbol{a}) - \hat{\mathcal{B}}^\pi \hat{Q}^k(\boldsymbol{s}, \boldsymbol{a})\right)^2\right], \tag{5.1.1}$$

In Theorem 5.1.1, we show that the resulting Q-function, $\hat{Q}^\pi \lim_{k\to\infty} \hat{Q}^k$, lower-bounds Q^π at all $\boldsymbol{s} \in \mathcal{D}, \boldsymbol{a} \in \mathcal{A}$. However, we can substantially tighten this bound if we are *only* interested in estimating $V^\pi(\boldsymbol{s})$. If we only require that the expected value of the $\hat{Q}^\pi$

under $\pi(\boldsymbol{a}|\boldsymbol{s})$ lower-bound V^{π}, we can improve this by introducing an additional Q-value *maximization* term on the data, $\pi_{\beta}(\boldsymbol{a}|\boldsymbol{s})$, resulting in (changes from Eq. 5.1.1 in red):

$$\hat{Q}^{k+1} \leftarrow \arg\min_{Q} \; \alpha \cdot \left(\mathbb{E}_{\boldsymbol{s}\sim\mathcal{D},\boldsymbol{a}\sim\mu(\boldsymbol{a}|\boldsymbol{s})}\left[Q(\boldsymbol{s},\boldsymbol{a})\right] - \mathbb{E}_{\boldsymbol{s}\sim\mathcal{D},\boldsymbol{a}\sim\hat{\pi}_{\beta}(\boldsymbol{a}|\boldsymbol{s})}\left[Q(\boldsymbol{s},\boldsymbol{a})\right] \right) + \frac{1}{2}\,\mathbb{E}_{\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}'\sim\mathcal{D}}\left[\left(Q(\boldsymbol{s},\boldsymbol{a}) - \hat{\mathcal{B}}^{\pi}\hat{Q}^{k}(\boldsymbol{s},\boldsymbol{a}) \right)^2 \right]. \quad (5.1.2)$$

In Theorem 5.1.2, we show that, while the resulting Q-value $\hat{Q}^{\pi}$ may not be a point-wise lower-bound, we have $\mathbb{E}_{\pi(\boldsymbol{a}|\boldsymbol{s})}[\hat{Q}^{\pi}(\boldsymbol{s},\boldsymbol{a})] \leq V^{\pi}(\boldsymbol{s})$ when $\mu(\boldsymbol{a}|\boldsymbol{s}) = \pi(\boldsymbol{a}|\boldsymbol{s})$. Intuitively, since Equation 5.1.2 maximizes Q-values under the behavior policy $\hat{\pi}_{\beta}$, Q-values for actions that are likely under $\hat{\pi}_{\beta}$ might be overestimated, and hence $\hat{Q}^{\pi}$ may not lower-bound Q^{π} pointwise. While in principle the maximization term can utilize other distributions besides $\hat{\pi}_{\beta}(\boldsymbol{a}|\boldsymbol{s})$, we prove in Appendix C.4.2 that the resulting value is not guaranteed to be a lower bound for other distribution besides $\hat{\pi}_{\beta}(\boldsymbol{a}|\boldsymbol{s})$, though other distributions besides $\hat{\pi}_{\beta}(\boldsymbol{a}|\boldsymbol{s})$ can still yield a lower bound if the Bellman error is also re-weighted to come from the distribution chosen to maximize the expected Q-value.

Theoretical Analysis

We first note that Equations 5.1.1 and 5.1.2 use the empirical Bellman operator, $\hat{\mathcal{B}}^{\pi}$, instead of the actual Bellman operator, $\mathcal{B}^{\pi}$. Following [254, 146, 251], we use concentration properties of $\hat{\mathcal{B}}^{\pi}$ to control this error. Formally, for all $\boldsymbol{s},\boldsymbol{a} \in \mathcal{D}$, with probability $\geq 1-\delta$, $|\hat{\mathcal{B}}^{\pi} - \mathcal{B}^{\pi}|(\boldsymbol{s},\boldsymbol{a}) \leq \frac{C_{r,T,\delta}}{\sqrt{|\mathcal{D}(\boldsymbol{s},\boldsymbol{a})|}}$, where $C_{r,T,\delta}$ is a constant dependent on the concentration properties (variance) of $r(\boldsymbol{s},\boldsymbol{a})$ and $T(\boldsymbol{s}'|\boldsymbol{s},\boldsymbol{a})$, and $\delta \in (0,1)$. For simplicity, we assume that $\hat{\pi}_{\beta}(\boldsymbol{a}|\boldsymbol{s}) > 0, \forall \boldsymbol{a} \in \mathcal{A}\,, \forall \boldsymbol{s} \in \mathcal{D}$. Let $\frac{1}{\sqrt{|\mathcal{D}|}}$ denote a vector of size $|\mathcal{S}||\mathcal{A}|$ containing square root inverse counts for each state-action pair, except when $\mathcal{D}(\boldsymbol{s},\boldsymbol{a}) = 0$, in which case the corresponding entry is a very large but finite value $\delta \geq \frac{2R_{\max}}{1-\gamma}$.

Theorem 5.1.1. *For any $\mu(\boldsymbol{a}|\boldsymbol{s})$ with $\mu \subset \hat{\pi}_{\beta}$, with probability $\geq 1-\delta$, $\hat{Q}^{\pi}$ (the Q-function obtained by iterating Equation 5.1.1) satisifies, $\forall \boldsymbol{s} \in \mathcal{D}, \boldsymbol{a}$:*

$$\hat{Q}^{\pi}(s,a) \leq Q^{\pi}(\boldsymbol{s},\boldsymbol{a}) - \alpha \left[(I - \gamma P^{\pi})^{-1} \frac{\mu}{\hat{\pi}_{\beta}} \right](\boldsymbol{s},\boldsymbol{a}) + \left[(I - \gamma P^{\pi})^{-1} \frac{C_{r,T,\delta} R_{\max}}{(1-\gamma)\sqrt{|\mathcal{D}|}} \right](\boldsymbol{s},\boldsymbol{a}).$$

Thus, if α is sufficiently large, then $\hat{Q}^{\pi}(\boldsymbol{s},\boldsymbol{a}) \leq Q^{\pi}(\boldsymbol{s},\boldsymbol{a}), \forall \boldsymbol{s} \in \mathcal{D}, \boldsymbol{a}$. When $\hat{\mathcal{B}}^{\pi} = \mathcal{B}^{\pi}$, any $\alpha > 0$ guarantees $\hat{Q}^{\pi}(\boldsymbol{s},\boldsymbol{a}) \leq Q^{\pi}(\boldsymbol{s},\boldsymbol{a}), \forall \boldsymbol{s} \in \mathcal{D}, \boldsymbol{a} \in \mathcal{A}$.

Next, we show that Equation 5.1.2 lower-bounds the expected value under the policy π, when $\mu = \pi$. We also show that Equation 5.1.2 does not lower-bound the Q-value estimates pointwise. For this result, we abuse notation and assume that $\frac{1}{\sqrt{|\mathcal{D}|}}$ refers to a

vector of inverse square root of only state counts, with a similar correction as before used to handle the entries of this vector at states with zero counts.

Theorem 5.1.2 (Equation 5.1.2 results in a tighter lower bound)**.** *The value of the policy under the Q-function from Equation 5.1.2, $\hat{V}^{\pi}(\boldsymbol{s}) = \mathbb{E}_{\pi(\boldsymbol{a}|\boldsymbol{s})}[\hat{Q}^{\pi}(\boldsymbol{s}, \boldsymbol{a})]$, lower-bounds the true value of the policy obtained via exact policy evaluation, $V^{\pi}(\boldsymbol{s}) = \mathbb{E}_{\pi(\boldsymbol{a}|\boldsymbol{s})}[Q^{\pi}(\boldsymbol{s}, \boldsymbol{a})]$, when $\mu = \pi$, according to:*

$$\hat{V}^{\pi}(\boldsymbol{s}) \leq V^{\pi}(\boldsymbol{s}) - \alpha \left[(I - \gamma P^{\pi})^{-1} \mathbb{E}_{\pi} \left[\frac{\pi}{\hat{\pi}_{\beta}} - 1 \right] \right] (\boldsymbol{s}) + \left[(I - \gamma P^{\pi})^{-1} \frac{C_{r,T,\delta} R_{\max}}{(1-\gamma)\sqrt{|\mathcal{D}|}} \right] (\boldsymbol{s}),$$

$\forall \boldsymbol{s} \in \mathcal{D}$. Thus, if $\alpha > \frac{C_{r,T} R_{\max}}{1-\gamma} \cdot \max_{\boldsymbol{s} \in \mathcal{D}} \frac{1}{|\sqrt{|\mathcal{D}(\boldsymbol{s})|}} \cdot \left[\sum_{\boldsymbol{a}} \pi(\boldsymbol{a}|\boldsymbol{s}) (\frac{\pi(\boldsymbol{a}|\boldsymbol{s})}{\hat{\pi}_{\beta}(\boldsymbol{a}|\boldsymbol{s})} - 1) \right]^{-1}$, $\forall \boldsymbol{s} \in \mathcal{D}$, $\hat{V}^{\pi}(\boldsymbol{s}) \leq V^{\pi}(\boldsymbol{s})$, with probability $\geq 1 - \delta$. When $\hat{\mathcal{B}}^{\pi} = \mathcal{B}^{\pi}$, then any $\alpha > 0$ guarantees $\hat{V}^{\pi}(\boldsymbol{s}) \leq V^{\pi}(\boldsymbol{s}), \forall \boldsymbol{s} \in \mathcal{D}$.

The analysis presented above assumes that no function approximation is used in the Q-function, meaning that each iterate can be represented exactly. We can further generalize the result in Theorem 5.1.2 to the case of both linear function approximators and non-linear neural network function approximators, where the latter builds on the neural tangent kernel (NTK) framework [145]. We present these results in Theorem C.4.1 and Theorem C.4.2 in Appendix C.4.1.

In summary, we showed that the basic CQL evaluation in Equation 5.1.1 learns a Q-function that lower-bounds the true Q-function Q^{π}, and the evaluation in Equation 5.1.2 provides a *tighter* lower bound on the expected Q-value of the policy π. For suitable α, both bounds hold under sampling error and function approximation. We also note that as more data becomes available and $|\mathcal{D}(\boldsymbol{s}, \boldsymbol{a})|$ increases, the theoretical value of α that is needed to guarantee a lower bound decreases, which indicates that in the limit of infinite data, a lower bound can be obtained by using extremely small values of α. Next, we will extend on this result into a complete RL algorithm.

5.1.3 *Conservative Q-Learning for Offline Policy Optimization*

We now present a general approach for model-free offline policy learning, which we refer to as conservative Q-learning (CQL). As discussed in Section 5.1.2, we can obtain Q-values that lower-bound the value of a policy π by solving Equation 5.1.2 with $\mu = \pi$. How should we utilize this for policy optimization? We could alternate between performing full off-policy evaluation for each policy iterate, $\hat{\pi}^k$, and one step of policy improvement. However, this can be expensive. Alternatively, since the policy $\hat{\pi}^k$ is typically derived from the Q-function, we could instead choose $\mu(\boldsymbol{a}|\boldsymbol{s})$ to approximate the policy that would maximize the current Q-function iterate, thus giving rise to an online algorithm.

We can formally capture such online algorithms by defining a family of optimization problems over $\mu(\boldsymbol{a}|\boldsymbol{s})$, presented below, with modifications from Equation 5.1.2 marked in red. An instance of this family is denoted by CQL($\mathcal{R}$) and is characterized by a particular choice of regularizer $\mathcal{R}(\mu)$:

$$\min_{Q} \max_{\mu} \; \alpha \left(\mathbb{E}_{\boldsymbol{s}\sim\mathcal{D},\boldsymbol{a}\sim\mu(\boldsymbol{a}|\boldsymbol{s})} \left[Q(\boldsymbol{s},\boldsymbol{a}) \right] - \mathbb{E}_{\boldsymbol{s}\sim\mathcal{D},\boldsymbol{a}\sim\hat{\pi}_{\beta}(\boldsymbol{a}|\boldsymbol{s})} \left[Q(\boldsymbol{s},\boldsymbol{a}) \right] \right) + \frac{1}{2} \mathbb{E}_{\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}'\sim\mathcal{D}} \left[\left(Q(\boldsymbol{s},\boldsymbol{a}) - \hat{\mathcal{B}}^{\pi_k} \hat{Q}^k(\boldsymbol{s},\boldsymbol{a}) \right)^2 \right] + \mathcal{R}(\mu) \quad (\mathrm{CQL}(\mathcal{R})) \,. \tag{5.1.3}$$

Variants of CQL

To demonstrate the generality of the CQL family of objectives, we discuss two specific instances within this family that are of special interest, and we evaluate them empirically in Section 4.5. If we choose $\mathcal{R}(\mu)$ to be the KL-divergence against a prior distribution, $\rho(\boldsymbol{a}|\boldsymbol{s})$, i.e., $\mathcal{R}(\mu) = -D_{\mathrm{KL}}(\mu, \rho)$, then we get $\mu(\boldsymbol{a}|\boldsymbol{s}) \propto \rho(\boldsymbol{a}|\boldsymbol{s}) \cdot \exp(Q(\boldsymbol{s},\boldsymbol{a}))$ (for a derivation, see Appendix C.1). Frist, if $\rho = \mathrm{Unif}(\boldsymbol{a})$, then the first term in Equation 5.1.3 corresponds to a soft-maximum of the Q-values at any state $\boldsymbol{s}$ and gives rise to the following variant of Equation 5.1.3, called CQL($\mathcal{H}$):

$$\min_{Q} \; \alpha \mathbb{E}_{\boldsymbol{s}\sim\mathcal{D}} \left[\log \sum_{\boldsymbol{a}} \exp(Q(\boldsymbol{s},\boldsymbol{a})) - \mathbb{E}_{\boldsymbol{a}\sim\hat{\pi}_{\beta}(\boldsymbol{a}|\boldsymbol{s})} \left[Q(\boldsymbol{s},\boldsymbol{a}) \right] \right] + \frac{1}{2} \mathbb{E}_{\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}'\sim\mathcal{D}} \left[\left(Q - \hat{\mathcal{B}}^{\pi_k} \hat{Q}^k \right)^2 \right]. \tag{5.1.4}$$

Second, if $\rho(\boldsymbol{a}|\boldsymbol{s})$ is chosen to be the previous policy $\hat{\pi}^{k-1}$, the first term in Equation 5.1.4 is replaced by an exponential weighted average of Q-values of actions from the chosen $\hat{\pi}^{k-1}(\boldsymbol{a}|\boldsymbol{s})$. Empirically, we find that this variant can be more stable with high-dimensional action spaces (e.g., Table 2) where it is challenging to estimate $\log \sum_{\boldsymbol{a}} \exp$ via sampling due to high variance. In Appendix C.1, we discuss an additional variant of CQL, drawing connections to distributionally robust optimization [245]. We will discuss a practical instantiation of a CQL deep RL algorithm in Section 5.1.5. CQL can be instantiated as either a Q-learning algorithm (with $\mathcal{B}^*$ instead of $\mathcal{B}^{\pi}$ in Equations 5.1.3, 5.1.4) or as an actor-critic algorithm.

Theoretical Analysis of CQL

Next, we will theoretically analyze CQL to show that the policy updates derived in this way are indeed "conservative", in the sense that each successive policy iterate is optimized against a lower bound on its value. For clarity, we state the results in the absence of finite-sample error, in this section, but sampling error can be incorporated in the same way as Theorems 5.1.1 and 5.1.2, and we discuss this in Appendix C.3. Theorem 5.1.3 shows that CQL($\mathcal{H}$) learns Q-value estimates that lower-bound the actual Q-function

under the action-distribution defined by the policy, π^k, under mild regularity conditions (slow updates on the policy).

Theorem 5.1.3 (CQL learns lower-bounded Q-values). *Let $\pi_{\hat{Q}^k}(\boldsymbol{a}|\boldsymbol{s}) \propto \exp(\hat{Q}^k(\boldsymbol{s},\boldsymbol{a}))$ and assume that $D_{\text{TV}}(\hat{\pi}^{k+1}, \pi_{\hat{Q}^k}) \leq \varepsilon$ (i.e., $\hat{\pi}^{k+1}$ changes slowly w.r.t to $\hat{Q}^k$). Then, the policy value under $\hat{Q}^k$, lower-bounds the actual policy value, $\hat{V}^{k+1}(\boldsymbol{s}) \leq V^{k+1}(\boldsymbol{s})\ \forall \boldsymbol{s} \in \mathcal{D}$ if*

$$\mathbb{E}_{\pi_{\hat{Q}^k}(\boldsymbol{a}|\boldsymbol{s})}\left[\frac{\pi_{\hat{Q}^k}(\boldsymbol{a}|\boldsymbol{s})}{\hat{\pi}_\beta(\boldsymbol{a}|\boldsymbol{s})} - 1\right] \geq \max_{\boldsymbol{a}\ s.t.\ \hat{\pi}_\beta(\boldsymbol{a}|\boldsymbol{s})>0}\left(\frac{\pi_{\hat{Q}^k}(\boldsymbol{a}|\boldsymbol{s})}{\hat{\pi}_\beta(\boldsymbol{a}|\boldsymbol{s})}\right) \cdot \varepsilon.$$

The LHS of this inequality is equal to the amount of conservatism induced in the value, $\hat{V}^{k+1}$ in iteration $k+1$ of the CQL update, if the learned policy were equal to soft-optimal policy for $\hat{Q}^k$, i.e., when $\hat{\pi}^{k+1} = \pi_{\hat{Q}^k}$. However, as the actual policy, $\hat{\pi}^{k+1}$, may be different, the RHS is the maximal amount of potential overestimation due to this difference. To get a lower bound, we require the amount of underestimation to be higher, which is obtained if ε is small, i.e. the policy changes slowly.

Our final result shows that CQL Q-function update is "gap-expanding", by which we mean that the difference in Q-values at in-distribution actions and over-optimistically erroneous out-of-distribution actions is higher than the corresponding difference under the actual Q-function. This implies that the policy $\pi^k(\boldsymbol{a}|\boldsymbol{s}) \propto \exp(\hat{Q}^k(\boldsymbol{s},\boldsymbol{a}))$, is constrained to be closer to the dataset distribution, $\hat{\pi}_\beta(\boldsymbol{a}|\boldsymbol{s})$, thus the CQL update implicitly prevents the detrimental effects of OOD action and distribution shift, which has been a major concern in offline RL settings [179, 95].

Theorem 5.1.4 (CQL is gap-expanding). *At iteration k, CQL expands the gap in expected Q-values under the data $\pi_\beta(\boldsymbol{a}|\boldsymbol{s})$ and μ_k, such that for large enough values of α_k, we have that $\forall \boldsymbol{s} \in \mathcal{D}$, $\mathbb{E}_{\pi_\beta(\boldsymbol{a}|\boldsymbol{s})}[\hat{Q}^k(\boldsymbol{s},\boldsymbol{a})] - \mathbb{E}_{\mu_k(\boldsymbol{a}|\boldsymbol{s})}[\hat{Q}^k(\boldsymbol{s},\boldsymbol{a})] > \mathbb{E}_{\pi_\beta(\boldsymbol{a}|\boldsymbol{s})}[Q^k(\boldsymbol{s},\boldsymbol{a})] - \mathbb{E}_{\mu_k(\boldsymbol{a}|\boldsymbol{s})}[Q^k(\boldsymbol{s},\boldsymbol{a})]$.*

When function approximation or sampling error makes OOD actions have higher learned Q-values, CQL backups are expected to be more robust, in that the policy is updated using Q-values that prefer in-distribution actions. As we will empirically show in Appendix C.2, prior offline RL methods that do not explicitly constrain or regularize the Q-function may not enjoy such robustness properties.

To summarize, we showed that the CQL algorithm learns lower-bound Q-values with large enough α, meaning that the final policy attains *at least* the estimated value. We also showed that the Q-function is *gap-expanding*, meaning that it should only ever *over-estimate* the gap between in-distribution and OOD actions, preventing OOD actions.

5.1.4 *Robust / Safe Policy Improvement Guarantees*

In Section 5.1.2 we proposed novel objectives for Q-function training such that the expected value of a policy under the resulting Q-function lower bounds the actual performance of the policy. In Section 5.1.3, we used the learned conservative Q-function for policy improvement. In this section, we show that this procedure actually optimizes a well-defined objective and provide a safe policy improvement result for CQL, along the lines of Theorems 1 and 2 in Laroche et al. [193].

To begin with, we define *empirical return* of any policy π, $J(\pi, \hat{M})$, which is equal to the discounted return of a policy π in the *empirical* MDP, $\hat{M}$, that is induced by the transitions observed in the dataset $\mathcal{D}$, i.e. $\hat{M} = \{s, a, r, s' \in \mathcal{D}\}$. $J(\pi, M)$ refers to the expected discounted return attained by a policy π in the actual underlying MDP, M. In Theorem 5.1.5, we first show that CQL (Equation 5.1.2) optimizes a well-defined penalized RL empirical objective. All proofs are found in Appendix C.4.4.

Theorem 5.1.5. *Let $\hat{Q}^\pi$ be the fixed point of Equation 5.1.2 in the tabular setting. Then the policy, π^* obtained by maximizing the expected values under the CQL Q-function can be expressed as: $\pi^*(\boldsymbol{a}|\boldsymbol{s}) \leftarrow \arg\max_\pi \; J(\pi, \hat{M}) - \alpha \frac{1}{1-\gamma} \mathbb{E}_{\boldsymbol{s} \sim d^\pi_{\hat{M}}(\boldsymbol{s})} \left[D_{CQL}(\pi, \hat{\pi}_\beta)(\boldsymbol{s}) \right]$, where $D_{CQL}(\pi, \pi_\beta)(\boldsymbol{s}) := \sum_{\boldsymbol{a}} \pi(\boldsymbol{a}|\boldsymbol{s}) \cdot \left(\frac{\pi(\boldsymbol{a}|\boldsymbol{s})}{\pi_\beta(\boldsymbol{a}|\boldsymbol{s})} - 1 \right)$.*

Intuitively, Theorem 5.1.5 says that CQL optimizes the return of a policy in the empirical MDP, $\hat{M}$, while also ensuring that the learned policy π is not too different from the behavior policy, $\hat{\pi}_\beta$ via a penalty that depends on D_{CQL}. Note that this penalty is implicitly introduced by virtue by the gap-expanding (Theorem 5.1.4) behavior of CQL. Next, building upon Theorem 5.1.5 and the analysis of CPO [3], we show that CQL provides a ζ-safe policy improvement over $\hat{\pi}_\beta$.

Theorem 5.1.6. *Let $\pi^*(\boldsymbol{a}|\boldsymbol{s})$ be the policy obtained in Theorem 5.1.5. Then, the policy $\pi^*(\boldsymbol{a}|\boldsymbol{s})$ is a ζ-safe policy improvement over $\hat{\pi}_\beta$ in the actual MDP M, i.e., $J(\pi^*, M) \geq J(\hat{\pi}_\beta, M) - \zeta$ with high probability $1 - \delta$, where $\zeta =$*

$$\mathcal{O}\left(\frac{\gamma}{(1-\gamma)^2}\right) \mathbb{E}_{\boldsymbol{s} \sim d^{\pi^*}_{\hat{M}}(\boldsymbol{s})} \left[\frac{\sqrt{|\mathcal{A}|}}{\sqrt{|\mathcal{D}(\boldsymbol{s})|}} \sqrt{D_{CQL}(\pi^*, \hat{\pi}_\beta)(\boldsymbol{s}) + 1} \right] - \underbrace{\left(J(\pi^*, \hat{M}) - J(\hat{\pi}_\beta, \hat{M})\right)}_{\geq \frac{\alpha}{1-\gamma} \mathbb{E}_{\boldsymbol{s} \sim d^{\pi^*}_{\hat{M}}(\boldsymbol{s})} \left[D_{CQL}(\pi^*, \hat{\pi}_\beta)(\boldsymbol{s}) \right]}$$

The expression of ζ in Theorem 5.1.6 consists of two terms: the first term captures the decrease in policy performance in M, that occurs due to the mismatch between $\hat{M}$ and M, also referred to as *sampling error*. The second term captures the increase in policy performance due to CQL in empirical MDP, $\hat{M}$. The policy π^* obtained by optimizing π

against the CQL Q-function improves upon the behavior policy, $\hat{\pi}_\beta$ for suitably chosen values of α. When sampling error is small, i.e., $|\mathcal{D}(\boldsymbol{s})|$ is large, then smaller values of α are enough to provide an improvement over the behavior policy.

To summarize, CQL optimizes a well-defined, penalized empirical RL objective, and performs high-confidence safe policy improvement over the behavior policy. The extent of improvement is negatively influenced by higher sampling error, which decays as more samples are observed.

5.1.5 *Practical Conservative Q-Learning Algorithm*

Algorithm 4 Conservative Q-Learning (both variants)

1: Initialize Q-function, Q_θ, and optionally a policy, π_ϕ.
2: **for all** step t in $\{1, \ldots, N\}$ **do**
3: Train the Q-function using G_Q gradient steps on objective from Equation 5.1.4
4: $\theta_t := \theta_{t-1} - \eta_Q \nabla_\theta \mathrm{CQL}(\mathcal{R})(\theta)$
5: (Use $\mathcal{B}^*$ for Q-learning, $\mathcal{B}^{\pi_{\phi_t}}$ for actor-critic)
6: (only with actor-critic) Improve policy π_ϕ via G_π gradient steps on ϕ with SAC-style entropy regularization:
7: $\phi_t := \phi_{t-1} + \eta_\pi \mathbb{E}_{\boldsymbol{s}\sim\mathcal{D}, \boldsymbol{a}\sim\pi_\phi(\cdot|\boldsymbol{s})}[Q_\theta(\boldsymbol{s},\boldsymbol{a}) - \log \pi_\phi(\boldsymbol{a}|\boldsymbol{s})]$
8: **end for**

We now describe two practical offline deep reinforcement learning methods based on CQL: an actor-critic variant and a Q-learning variant. Pseudocode is shown in Algorithm 7, with differences from conventional actor-critic algorithms (e.g., SAC [122]) and deep Q-learning algorithms (e.g., DQN [232]) in red. Our algorithm uses the CQL($\mathcal{H}$) (or CQL($\mathcal{R}$) in general) objective from the CQL framework for training the Q-function Q_θ, which is parameterized by a neural network with parameters θ. For the actor-critic algorithm, a policy π_ϕ is trained as well. Our algorithm modifies the objective for the Q-function (swaps out Bellman error with CQL($\mathcal{H}$)) or CQL(ρ) in a standard actor-critic or Q-learning setting, as shown in Line 3. As discussed in Section 5.1.3, due to the explicit penalty on the Q-function, CQL methods do not use a policy constraint, unlike prior offline RL methods [179, 343, 296, 204]. Hence, we do not require fitting an additional behavior policy estimator, simplifying our method.

Implementation details. Our algorithm requires an addition of only **20** lines of code on top of standard implementations of soft actor-critic (SAC) [122] for continuous control experiments and on top of QR-DQN [51] for the discrete control. The tradeoff factor, α is is fixed at constant values described in Appendix C.6 for gym tasks and discrete control and is automatically tuned via Lagrangian dual gradient descent for other domains. We use default hyperparameters from SAC, except that the learning rate for the policy was

chosen from {3e-5, 1e-4, 3e-4}, and is less than or equal to the Q-function, as dictated by Theorem 5.1.3. Elaborate details are provided in Appendix C.6.

5.1.6 *Related Work*

We now briefly discuss prior work in offline RL and off-policy evaluation, comparing and contrasting these works with our approach.

Off-policy evaluation (OPE). Several different paradigms have been used to perform off-policy evaluation. Earlier works [268, 260, 269] used per-action importance sampling on Monte-Carlo returns to obtain an OPE return estimator. Recent approaches [217, 102, 237, 373] use marginalized importance sampling by directly estimating the state-distribution importance ratios via some form of dynamic programming [204] and typically exhibit less variance than per-action importance sampling at the cost of bias. Because these methods use dynamic programming, they can suffer from OOD actions [204, 102, 128, 237]. In contrast, the regularizer in CQL explicitly addresses the impact of OOD actions due to its gap-expanding behavior, and obtains conservative value estimates.

Offline RL. Prior methods have attempted to solve the offline RL problem by constraining the learned policy to be "close" to the behavior policy, for example as measured by KL-divergence [149, 343, 259, 296], Wasserstein distance [343], or MMD [179]. Most of these methods require a separately estimated model to the behavior policy, $\pi_\beta(\boldsymbol{a}|\boldsymbol{s})$ [95, 179, 343, 149, 296, 299], and are thus limited by their ability to accurately estimate the unknown behavior policy [241], which might be especially complex in settings where the data is collected from multiple sources [204]. In contrast, CQL does not require estimating the behavior policy. Prior work has explored some forms of Q-function penalties [138, 331], but only in the standard online RL setting with demonstrations. Luo et al. [219] learn a conservatively-extrapolated value function by enforcing a linear extrapolation property over the state-space, and a learned dynamics model to obtain policies for goal-reaching tasks. Kakade and Langford [157] proposed the CPI algorithm, that improves a policy conservatively in online RL.

Alternate prior approaches to offline RL estimate some sort of uncertainty to determine the trustworthiness of a Q-value prediction [179, 6, 204], typically using uncertainty estimation techniques from exploration in online RL [254, 146, 253, 36]. These methods have not been generally performant in offline RL [95, 179, 204] due to the high-fidelity requirements of uncertainty estimates in offline RL [204]. Robust MDPs [144, 261, 316, 248] have been a popular theoretical abstraction for offline RL, but tend to be highly conservative in policy improvement. We expect CQL to be less conservative since CQL does not underestimate Q-values for all state-action tuples. Works on high confidence policy improvement [322] provides safety guarantees for improvement but tend to be conservative. The gap-

expanding property of CQL (Theorem 5.1.4), is related to how gap-increasing Bellman backup operators [23, 218] are more robust to estimation error in online RL.

Theoretical results. Our theoretical results (Theorems 5.1.5, 5.1.6) are related to prior work on safe policy improvement [193, 261], and a direct comparison to Theorems 1 and 2 in Laroche et al. [193] suggests similar quadratic dependence on the horizon and an inverse square-root dependence on the counts. Our bounds improve over the ∞-norm bounds in Petrik et al. [261].

5.1.7 *Experimental Evaluation of CQL*

We compare CQL to prior offline RL methods on a range of domains and dataset compositions, including continuous and discrete action spaces, state observations of varying dimensionality, and high-dimensional image inputs. We first evaluate actor-critic CQL, using CQL($\mathcal{H}$) from Algorithm 7, on continuous control datasets from the D4RL benchmark [86]. We compare to: prior offline RL methods that use a policy constraint – BEAR [179] and BRAC [343]; SAC [122], an off-policy actor-critic method that we adapt to offline setting; and behavioral cloning (BC). The code and instructions for reproducing our results can be found at: https://github.com/young-geng/JaxCQL. Please refer to this repository and the latest D4RL for future comparisons. D4RL-v0 is deprecated.

Task Name	SAC	BC	BEAR	BRAC-p	BRAC-v	CQL
halfcheetah-medium-v2	-4.3	42.6	38.6	44.0	51	48.4 $\pm$ 0.3
walker2d-medium-v2	0.9	75.3	33.2	72.7	81.3	82.0 $\pm$ 1.0
hopper-medium-v2	0.8	52.9	47.6	31.2	86.6	71.8 $\pm$ 2.5
halfcheetah-medium-expert-v2	1.8	55.2	51.7	43.8	44.0	87.3 $\pm$ 0.2
walker2d-medium-expert-v2	1.9	52.5	10.8	-0.3	111.6	106.1 $\pm$ 7.2
hopper-medium-expert-v2	1.6	107.5	4.0	1.1	79.0	109.2 $\pm$ 3.6
halfcheetah-medium-replay-v2	-2.4	36.6	36.2	47.6	45.3	47.0 $\pm$ 0.2
hopper-medium-replay-v2	3.5	18.1	25.3	0.7	96.2	93.8 $\pm$ 2.8
walker2d-medium-replay-v2	1.9	26.0	10.8	-0.3	85.0	86.2 $\pm$ 0.2
Total	5.7	466.7	-	-	680.0	**731.8**

Table 1: Performance of CQL and prior methods on gym domains from D4RL, on the normalized return metric, averaged over 3 seeds. Note that CQL performs similarly or better than the best prior method with simple datasets, and outperforms prior methods with complex distributions ("–medium-replay", "–medium-expert").

Gym domains. Results for the gym domains are shown in Table 1. The results for BEAR, BRAC, SAC, and BC are based on numbers reported by Fu et al. [86]. We find that across the board, on both the datasets generated by a single mediocre policy, marked as "-medium" and the datasets generated by mixing together experience from multiple policies ("-medium-replay" and "-medium-expert"), that are more likely to be encountered in practical problems, CQL outperforms prior methods.

Adroit tasks. The more complex Adroit [274] tasks in D4RL require controlling a 24-DoF robotic hand, using limited data from human demonstrations. These tasks are substantially more difficult than the gym tasks in terms of both the dataset composition and high dimensionality. Prior offline RL methods generally struggle to learn meaningful behaviors on these tasks, and the strongest baseline is BC. As shown in Table 2, CQL variants are the only methods that improve over BC, attaining scores that are **2-9x** those of the next best offline RL method. CQL(ρ) with $\rho = \hat{\pi}^{k-1}$ (the previous policy) outperforms CQL($\mathcal{H}$) on a number of these tasks, due to the higher action dimensionality resulting in higher variance for the CQL($\mathcal{H}$) importance weights.

Domain	Task Name	BC	SAC	BEAR	BRAC-p	BRAC-v	CQL($\mathcal{H}$)	CQL(ρ)
AntMaze	antmaze-umaze-v2	54.6	0.0	73.0	50.0	70.0	**94.0**	-
	antmaze-umaze-diverse-v2	45.6	0.0	61.0	40.0	70.0	47.3	-
	antmaze-medium-play-v2	0.0	0.0	0.0	0.0	0.0	**62.4**	-
	antmaze-medium-diverse-v2	0.0	0.0	8.0	0.0	0.0	**74.3**	-
	antmaze-large-play-v2	0.0	0.0	0.0	0.0	0.0	**34.2**	-
	antmaze-large-diverse-v2	0.0	0.0	0.0	0.0	0.0	**40.7**	-
	Total (antmazes)	100.2	0.0	-	-	-	**352.9**	-
Adroit	pen-human	34.4	6.3	-1.0	8.1	0.6	37.5	**55.8**
	hammer-human	1.5	0.5	0.3	0.3	0.2	**4.4**	2.1
	door-human	0.5	3.9	-0.3	-0.3	-0.3	**9.9**	**9.1**
	relocate-human	0.0	0.0	-0.3	-0.3	-0.3	0.20	**0.35**
	pen-cloned	**56.9**	23.5	26.5	1.6	-2.5	39.2	40.3
	hammer-cloned	0.8	0.2	0.3	0.3	0.3	2.1	**5.7**
	door-cloned	-0.1	0.0	-0.1	-0.1	-0.1	0.4	**3.5**
	relocate-cloned	**-0.1**	-0.2	-0.3	-0.3	-0.3	**-0.1**	**-0.1**
Kitchen	kitchen-complete	33.8	15.0	0.0	0.0	0.0	**43.8**	31.3
	kitchen-partial	33.8	0.0	13.1	0.0	0.0	**49.8**	**50.1**
	kitchen-undirected	47.5	2.5	47.2	0.0	0.0	**51.0**	**52.4**

Table 2: Normalized scores of all methods on AntMaze, Adroit, and kitchen domains from D4RL, averaged across 4 seeds. On the harder mazes, CQL is the *only* method that attains non-zero returns, and is the only method to outperform simple behavioral cloning on Adroit tasks with human demonstrations. We observed that the CQL(ρ) variant, which avoids importance weights, trains more stably, with no sudden fluctuations in policy performance over the course of training, on the higher-dimensional Adroit tasks.

AntMaze. These tasks require composing parts of suboptimal trajectories to form more optimal policies for reaching goals on a MuJoco Ant robot. Prior methods make some progress on the simpler U-maze, but only CQL is able to make meaningful progress on the much harder medium and large mazes, outperforming prior methods.

Kitchen tasks. Next, we evaluate CQL on the Franka kitchen domain [119] from D4RL [91]. The goal is to control a 9-DoF robot to manipulate multiple objects (microwave, kettle, etc.) *sequentially*, in a single episode to reach a desired configuration, with only sparse 0-1 completion reward for every object that attains the target configuration. These tasks are especially challenging, since they require composing parts of trajectories, precise long-horizon manipulation, and handling human-provided teleoperation data. As shown

in Table 2, CQL outperforms prior methods in this setting, and is the only method that outperforms behavioral cloning, attaining over **40%** success rate on all tasks.

Offline RL on Atari games[1]**.** Lastly, we evaluate a discrete-action Q-learning variant of CQL (Algorithm 7) on offline, image-based Atari games [22]. We compare CQL to REM [6] and QR-DQN [51] on the five Atari tasks (Pong, Breakout, Qbert, Seaquest and Asterix) that are evaluated in detail by Agarwal et al. [6], using the dataset released by the authors.

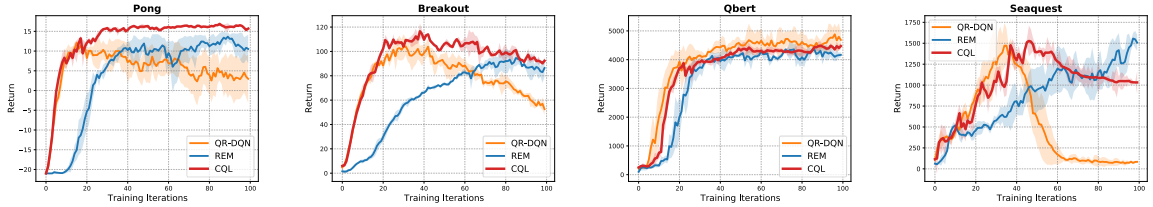


Figure 10: Performance of CQL, QR-DQN and REM as a function of training steps (x-axis) in setting **(1)** when provided with only the first 20% of the samples of an online DQN run. Note that CQL is able to learn stably on 3 out of 4 games, and its performance does not degrade as steeply as QR-DQN on Seaquest.

Following the evaluation protocol of Agarwal et al. [6], we evaluated on two types of datasets, both of which were generated from the DQN-replay dataset, released by [6]: **(1)** a dataset consisting of the first 20% of the samples observed by an online DQN agent and **(2)** datasets consisting of only 1% and 10% of all samples observed by an online DQN agent (Figures 6 and 7 in [6]). In setting **(1)**, shown in Figure 10, CQL generally achieves similar or better performance throughout as QR-DQN and REM. When only using only 1% or 10% of the data, in setting **(2)** (Table 3), CQL substantially outperforms REM and QR-DQN, especially in the harder 1% condition, achieving **36x** and **6x** times the return of the best prior method on Q*bert and Breakout, respectively.

Analysis of CQL. Finally, we perform empirical evaluation to verify that CQL indeed lower-bounds the value function, thus verifying Theorems 5.1.2, Appendix C.4.1 empirically. To this end, we estimate the average value of the learned policy predicted by CQL, $\mathbb{E}_{\mathbf{s}\sim\mathcal{D}}[\hat{V}^k(\mathbf{s})]$, and report the difference against the actual discounted return of the policy π^k in Table 4. We also estimate these values for baselines, including the minimum predicted Q-value under an ensemble [122, 97] of Q-functions with varying ensemble sizes, which is a standard technique to prevent overestimed Q-values [97, 122, 129] and BEAR [179], a policy constraint method. The results show that CQL learns a lower bound for all three tasks, whereas the baselines are prone to overestimation. We also evaluate a variant of CQL that uses Equation 5.1.1, and observe that the resulting values are lower (that is, underestimate the true values) as compared to CQL($\mathcal{H}$). This provides empirical evidence that CQL($\mathcal{H}$) attains a tighter lower bound than the point-wise bound

[1] Results for CQL on more Atari games, with varied dataset compositions can be found in Appendix F of DR3 (Kumar et al. ICLR 2022), available at the following arXiv URL: https://arxiv.org/abs/2112.04716

Task Name	QR-DQN	REM	CQL($\mathcal{H}$)
Pong (1%)	-13.8	-6.9	**19.3**
Breakout	7.9	11.0	**61.1**
Q*bert	383.6	343.4	**14012.0**
Seaquest	672.9	499.8	**779.4**
Asterix	166.3	386.5	**592.4**
Pong (10%)	15.1	8.9	**18.5**
Breakout	151.2	86.7	**269.3**
Q*bert	7091.3	8624.3	**13855.6**
Seaquest	2984.8	**3936.6**	3674.1
Asterix	**189.2**	75.1	156.3

Table 3: CQL, REM and QR-DQN in setting **(1)** with 1% data (top), and 10% data (bottom). CQL outperforms prior methods with 1% data, and usually attains better performance with 10% data.

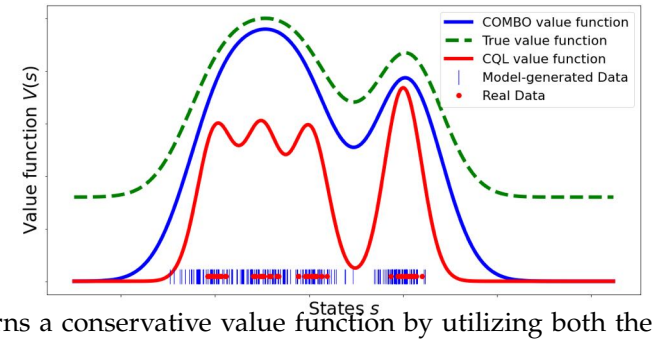


Figure 11: COMBO learns a conservative value function by utilizing both the offline dataset as well as simulated data from the model. Crucially, COMBO does not require uncertainty quantification, and the value function learned by COMBO is a tighter lower-bound of the true value compared to CQL. This enables COMBO to steer the agent towards higher value states compared to CQL, which may steer towards sub-optimal states as illustrated in the figure.

in Equation 5.1.1, as per Theorem 5.1.2. We also present an empirical analysis to show that Theorem 5.1.4, that CQL is gap-expanding, holds in practice in Appendix C.2.

Task Name	CQL($\mathcal{H}$)	CQL (Eqn. 5.1.1)	Ensemble(2)	Ens.(4)	Ens.(10)	Ens.(20)	BEAR
hopper-medium-expert	**-43.20**	-151.36	3.71e6	2.93e6	0.32e6	24.05e3	65.93
hopper-mixed	**-10.93**	-22.87	15.00e6	59.93e3	8.92e3	2.47e3	1399.46
hopper-medium	**-7.48**	-156.70	26.03e12	437.57e6	1.12e12	885e3	4.32

Table 4: Difference between predicted policy values and the true policy value for CQL, a variant of CQL that uses Equation 5.1.1, the minimum of an ensemble of varying sizes, and BEAR [179] on three D4RL datasets. CQL is the only method that lower-bounds the actual return (i.e., has negative differences), and CQL($\mathcal{H}$) is much less conservative than CQL (Eqn. 5.1.1).

5.2 Model-based Conservative Value Function Estimation

In Section 5.1, we discussed conservative Q-learning, a model-free algorithm that learns conservative value functions on offline datasets. An alternative way to use offline data is to first attempt to estimate the dynamics of the underlying task, followed by training control policies via planning or explicit policy optimization. Empirically, in the standard RL

problem setting, model-based RL algorithms have achieved better generalization [313, 147], especially in multi-task settings [365]. Specifically, one performant recipe for model-based RL is to utilize an estimated or learned dynamics model to generate data for a downstream model-free control pipeline, such as one based on Q-learning [147].

Despite appealing properties of model-based RL, existing model-based methods still suffer from similar problems in the offline setting: any learned model of the environment dynamics is bound to make erroneous predictions on states or actions that are unlikely to appear within the support of the training dataset, and inevitably, downstream policy optimization will exploit these errors to produce policies that appear performant under the learned model, but fail to act reliably when deployed. To address this issue, existing offline model-based RL algorithms [166, 365] rely on precise quantifying uncertainty in predictions made by the learned dynamics model. However, as we show in Section 5.2.2, obtaining calibrated uncertainty estimates can be extremely difficult, especially for complex datasets with deep network models.

To alleviate the need for calibrated uncertainty estimation, in this section, we develop a new model-based offline RL algorithm, building on conservative value estimation paradigm. Our approach, conservative offline model-based policy optimization (COMBO), retains the benefits of model-based algorithms while removing the reliance on uncertainty estimation. COMBO first learns a dynamics model using the offline dataset. Subsequently, it employs an actor-critic method where the value function is learned using both the offline dataset as well as synthetically generated data from the model. Crucially, this value function is trained via a variant of conservative optimization, that penalizes the value function on state-action tuples that are visited by the policy by simulating the learned model, but are not in the support of the offline dataset. Using the theoretical tools from Section 5.1, we theoretically show that for any policy, the Q-function learned with COMBO is a lower bound of the true Q-function, making it a good surrogate for policy optimization. We also theoretically show that the Q-function learned by COMBO represents a tighter lower bound of the true Q-function when the model bias is low compared to prior model-free algorithms like CQL [181]. Thus, as a consequence of optimizing a tighter lower bound, COMBO has the potential to learn higher rewarding policies compared to prior model-free algorithms. This is illustrated in Figure 11.

Finally, in our experiments, we find that COMBO matches or exceeds the state-of-the-art results in commonly studied benchmark tasks for offline RL. Specifically, COMBO achieves the highest score in 9 out of 12 continuous control domains from the D4RL [92] benchmark suite, while the next best algorithm achieves the highest score in only 3 out of the 12 domains. We also show that COMBO achieves the best performance in tasks that require out-of-distribution generalization and outperforms previous offline model-based RL methods on this image-based robotic manipulation task.

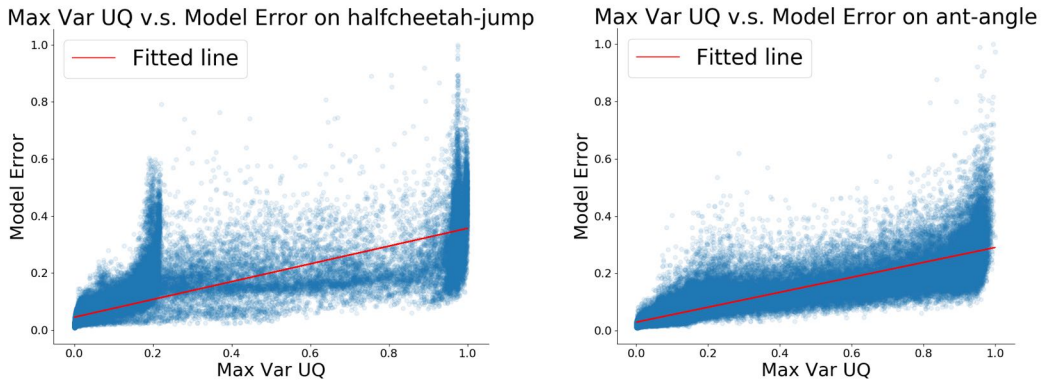


Figure 12: We visualize the fitted linear regression line between the model error and two uncertainty quantification methods maximum learned variance over the ensemble (denoted as **Max Var**) on two tasks that test the generalization abilities of offline RL algorithms (halfcheetah-jump and ant-angle). We show that **Max Var** struggles to predict the true model error. Such visualizations indicates that uncertainty quantification is challenging with deep neural networks and could lead to poor performance in model-based offline RL in settings where out-of-distribution generalization is needed. In the meantime, COMBO addresses this issue by removing the burden of performing uncertainty quantification.

5.2.1 *Additional Notation for Model-Based RL*

Typical model-based offline RL algorithms use the given dataset $\mathcal{D}$ to train a dynamics model $\widehat{T}$ using maximum likelihood estimation as: $\min_{\widehat{T}} \mathbb{E}_{(\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}')\sim\mathcal{D}}\left[\log \widehat{T}(\boldsymbol{s}'|\boldsymbol{s},\boldsymbol{a})\right]$. A reward model $\hat{r}(\boldsymbol{s},\boldsymbol{a})$ can also be learned similarly. Once a model has been learned, we can construct the learned MDP $\widehat{\mathcal{M}} = (\mathcal{S}, \mathcal{A}, \widehat{T}, \hat{r}, \mu_0, \gamma)$, which has the same state and action spaces, but uses the learned dynamics and reward function.

We will use $\mathcal{D}_{\text{model}}$ is a dataset obtained by simulating the current policy using the learned dynamics model. Specifically, at each iteration, MBPO [147], a model-based online RL algorithm that we build off of performs k-step rollouts using $\widehat{T}$ starting from state $\boldsymbol{s} \in \mathcal{D}$ with a particular rollout policy $\mu(\boldsymbol{a}|\boldsymbol{s})$, adds the model-generated data to $\mathcal{D}_{\text{model}}$, and optimizes the policy with a batch of data sampled from $\mathcal{D} \cup \mathcal{D}_{\text{model}}$ where each datapoint in the batch is drawn from $\mathcal{D}$ with probability $f \in [0, 1]$ and $\mathcal{D}_{\text{model}}$ with probability $1 - f$.

5.2.2 *Analysis: Uncertainty Calibration in Offline Model-Based RL*

In this section, we first analyze the efficacy of uncertainty estimation in existing model-based offline RL approaches. Specifically, we perform empirical evaluations to study whether uncertainty quantification with deep neural networks, especially in the setting of dynamics model learning, is challenging and could cause problems with uncertainty-based model-based offline RL methods such as MOReL [166] and MOPO [365]. We pick tasks where the underlying model-based offline RL method is required to produce policies that need to go further away from the data distribution to obtain good performance. In

our evaluations, we consider maximum learned variance over the ensemble (denoted as **Max Var**) $\max_{i=1,\ldots,N} \|\Sigma_\theta^i(\boldsymbol{s}, \boldsymbol{a})\|_{\mathrm{F}}$ (used in MOPO).

We consider two tasks halfcheetah-jump and ant-angle (more details about the task in Appendix C.8.4). We normalize both the model prediction error and the uncertainty estimates to be within scale $[0, 1]$ and performs linear regression that learns the mapping between the uncertainty estimates and the true model error. As shown in Figure 12, on both tasks, **Max Var** is unable to accurately predict the true model error, suggesting that uncertainty estimation used by offline model-based methods is not accurate and might be the major factor that results in its poor performance. On the other hand, our approach, COMBO, alleviates the need for accurate uncertainty quantification by viewing model-based offline RL from the lens of conservative value function learning.

5.2.3 *Conservative Offline Model-Based Policy Optimization*

Our goal in this section is to develop a model-based offline RL algorithm that enables optimizing a lower bound on the policy performance, but without requiring uncertainty quantification. We achieve this by extending conservative Q-learning, from the previous section, into the model-based setting. Our algorithm, summarized in Algorithm 5, alternates between a conservative policy evaluation step and a policy improvement step, which we outline below.

Conservative Policy Evaluation: Given a policy π, an offline dataset $\mathcal{D}$, and a learned model of the MDP $\hat{\mathcal{M}}$, the goal in this step is to obtain a conservative estimate of Q^π. To achieve this, we penalize the Q-values evaluated on data drawn from a particular state-action distribution that is more likely to be out-of-support while pushing up the Q-values on state-action pairs that are trustworthy, which is implemented by repeating the following recursion:

$$\min_Q \beta \left(\mathbb{E}_{\boldsymbol{s},\boldsymbol{a}\sim\rho(\boldsymbol{s},\boldsymbol{a})}[Q(\boldsymbol{s},\boldsymbol{a})] - \mathbb{E}_{\boldsymbol{s},\boldsymbol{a}\sim\mathcal{D}}\left[Q(\boldsymbol{s},\boldsymbol{a})\right] \right) + \frac{1}{2}\mathbb{E}_{\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}'\sim d_f}\left[\left(Q(\boldsymbol{s},\boldsymbol{a}) - \widehat{\mathcal{B}}^\pi \hat{Q}^k(\boldsymbol{s},\boldsymbol{a}) \right)^2 \right]. \quad (5.2.1)$$

Here, $\rho(\boldsymbol{s}, \boldsymbol{a})$ and d_f are sampling distributions that we can choose. Model-based algorithms allow ample flexibility for these choices while providing the ability to control the bias introduced by these choices. For $\rho(\boldsymbol{s}, \boldsymbol{a})$, we make the following choice:

$$\rho(\boldsymbol{s}, \boldsymbol{a}) = d^\pi_{\widehat{\mathcal{M}}}(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s}),$$

where $d^\pi_{\widehat{\mathcal{M}}}(\boldsymbol{s})$ is the discounted marginal state distribution when executing π in the learned model $\widehat{\mathcal{M}}$. Samples from $d^\pi_{\widehat{\mathcal{M}}}(\boldsymbol{s})$ can be obtained by rolling out π in $\widehat{\mathcal{M}}$. Similarly, d_f is an $f-$interpolation between the offline dataset and rollouts from the model:

$$d^\mu_f(\boldsymbol{s}, \boldsymbol{a}) := f\ d(\boldsymbol{s}, \boldsymbol{a}) + (1 - f)\ d^\mu_{\widehat{\mathcal{M}}}(\boldsymbol{s}, \boldsymbol{a}),$$

Algorithm 5 COMBO: Conservative Model Based Offline Policy Optimization

Require: Offline dataset $\mathcal{D}$, rollout distribution $\mu(\cdot|\boldsymbol{s})$, learned dynamics model $\widehat{T}_\theta$, initialized policy and critic π_ϕ and Q_ψ.

1: Train the probabilistic dynamics model $\widehat{T}_\theta(\boldsymbol{s}', r|\boldsymbol{s}, \boldsymbol{a}) = \mathcal{N}(\mu_\theta(\boldsymbol{s}, \boldsymbol{a}), \Sigma_\theta(\boldsymbol{s}, \boldsymbol{a}))$ on $\mathcal{D}$.
2: Initialize the replay buffer $\mathcal{D}_{\text{model}} \leftarrow \varnothing$.
3: **for** $i = 1, 2, 3, \cdots$, **do**
4: Perform rollouts by drawing samples from μ and $\widehat{T}_\theta$ starting from states in $\mathcal{D}$. Add model rollouts to $\mathcal{D}_{\text{model}}$.
5: Conservatively evaluate the policy by solving eq. 5.2.1 to obtain $\hat{Q}_\psi^{\pi_\phi^i}$ using data sampled from $\mathcal{D} \cup \mathcal{D}_{\text{model}}$.
6: Improve policy under state marginal of d_f by solving eq. 5.2.2 to obtain π_ϕ^{i+1}.
7: **end for**

where $f \in [0, 1]$ is the ratio of the datapoints drawn from the offline dataset as defined in Section 5.2.1 and $\mu(\cdot|\boldsymbol{s})$ is the rollout distribution used with the model, which can be modeled as π or a uniform distribution. For brevity, we also denote $d_f := d_f^\mu$.

Under such choices of ρ and d_f, we push down Q-values on state-action tuples from model rollouts and push up Q-values on the real state-action pairs from the offline dataset. When updating Q-values with the Bellman backup, we use a mixture of both the model-generated data and the real data, similar to Dyna [313]. Note that in comparison to CQL and other model-free algorithms, COMBO learns the Q-function over a richer set of states beyond the states in the offline dataset. This is made possible by performing rollouts under the learned dynamics model, denoted by $d^\mu_{\widehat{\mathcal{M}}}(\boldsymbol{s}, \boldsymbol{a})$. We will show in Section 5.2.4 that the Q function learned by repeating the recursion in Equation 5.2.1 provides a lower bound on the true Q function, without the need for explicit uncertainty estimation. Furthermore, we will theoretically study the advantages of using synthetic data from the learned model, and characterize the impacts of model bias.

Policy improvement using a conservative critic: After learning a conservative critic $\hat{Q}^\pi$, we improve the policy as:

$$\pi' \leftarrow \arg\max_\pi \; \mathbb{E}_{\boldsymbol{s}\sim\rho, \boldsymbol{a}\sim\pi(\cdot|\boldsymbol{s})} \left[\hat{Q}^\pi(\boldsymbol{s}, \boldsymbol{a})\right] \tag{5.2.2}$$

where $\rho(\boldsymbol{s})$ is the state marginal of $\rho(\boldsymbol{s}, \boldsymbol{a})$. When policies are parameterized with neural networks, we approximate the arg max with a few steps of gradient descent. In addition, entropy regularization can also be used to prevent the policy from becoming degenerate if required [126]. In Section 5.2.4.2, we show that the resulting policy is guaranteed to improve over the behavior policy.

Practical implementation details. Our practical implementation largely follows MOPO, with the key exception that we perform conservative policy evaluation as outlined in

this section, rather than using uncertainty-based reward penalties. Following MOPO, we represent the probabilistic dynamics model using a neural network, with parameters θ, that produces a Gaussian distribution over the next state and reward: $\widehat{T}_\theta(\boldsymbol{s}_{t+1}, r|\boldsymbol{s}, \boldsymbol{a}) = \mathcal{N}(\mu_\theta(\boldsymbol{s}_t, \boldsymbol{a}_t), \Sigma_\theta(\boldsymbol{s}_t, \boldsymbol{a}_t))$. The model is trained via maximum likelihood. For training the conservative critic, which is the major distinction between COMBO and MOPO, the fixed constant β is tuned with an offline cross-validation scheme for all low-dimensional continuous control tasks and is decided with a limited number of rollouts in the actual environment in the vision-based environments. We set the ratio $f = 0.5$ to have an equal split between model rollouts and data from the offline dataset. For conservative policy evaluation (eq. 5.2.1) and policy improvement (eq. 5.2.2), we augment ρ with states sampled from the offline dataset, which shows more stable improvement in practice. Additional details about practical implementation are provided in Appendix C.8.1.

5.2.4 *Theoretical Analysis of COMBO*

In this section, we theoretically analyze a simplified variant of COMBO and show that, like CQL, it continues to optimize a lower-bound on the expected return of the learned policy. The specific simplification we consider is a Bellman backup operator that considers a mixture of the model-free and model-based Bellman backup operator, with a weight f. The lower bound for COMBO that we show is close to the actual policy performance (modulo sampling error) when the policy's state-action marginal distribution is in support of the state-action marginal of the behavior policy and conservatively estimates the performance of a policy otherwise. By optimizing the policy against this lower bound, COMBO guarantees policy improvement beyond the behavior policy. Furthermore, we use these insights to discuss cases when COMBO is less conservative compared to CQL.

5.2.4.1 *COMBO Optimizes a Lower Bound*

We first show that training the Q-function using Equation 5.2.1 produces a Q-function such that the expected off-policy policy improvement objective [58] computed using this learned Q-function lower-bounds its actual value. We will reuse notation for d_f and d from Section 5.2.3. Assuming that the Q-function is tabular, the Q-function found by approximate dynamic programming in iteration k, can be obtained by differentiating Equation 5.2.1 with respect to Q^k (see App. C.7 for details):

$$\hat{Q}^{k+1}(\boldsymbol{s}, \boldsymbol{a}) = (\widehat{\mathcal{B}}^\pi Q^k)(\boldsymbol{s}, \boldsymbol{a}) - \beta \frac{\rho(\boldsymbol{s}, \boldsymbol{a}) - d(\boldsymbol{s}, \boldsymbol{a})}{d_f(\boldsymbol{s}, \boldsymbol{a})}. \tag{5.2.3}$$

Equation 5.2.3 effectively applies a penalty that depends on the three distributions appearing in the COMBO critic training objective (Equation 5.2.1), of which ρ and d_f are

free variables that we choose in practice as discussed in Section 5.2.3. For a given iteration k of Equation 5.2.3, we further define the expected penalty under $\rho(\boldsymbol{s}, \boldsymbol{a})$ as:

$$\nu(\rho, f) := \mathbb{E}_{\boldsymbol{s},\boldsymbol{a}\sim\rho(\boldsymbol{s},\boldsymbol{a})}\left[\frac{\rho(\boldsymbol{s}, \boldsymbol{a}) - d(\boldsymbol{s}, \boldsymbol{a})}{d_f(\boldsymbol{s}, \boldsymbol{a})}\right]. \tag{5.2.4}$$

Next, we will show that the Q-function learned by COMBO lower-bounds the actual Q-function under the initial state distribution μ_0 and any policy π. We also show that the asymptotic Q-function learned by COMBO lower-bounds the actual Q-function of any policy π with high probability for a large enough $\beta \geq 0$, which we include in Appendix C.7.2. Let $\overline{\mathcal{M}}$ represent the empirical MDP which uses the empirical transition model based on raw data counts. The Bellman backups over the dataset distribution d_f in equation 5.2.1 can be interpreted as an $f-$interpolation of the backup operator in the empirical MDP (denoted by $\mathcal{B}^{\pi}_{\overline{\mathcal{M}}}$) and the backup operator under the learned model $\widehat{\mathcal{M}}$ (denoted by $\mathcal{B}^{\pi}_{\widehat{\mathcal{M}}}$). The empirical backup operator suffers from sampling error, but is unbiased in expectation, whereas the model backup operator induces bias but no sampling error. We assume that all of these backups enjoy concentration properties with concentration coefficient $C_{r,T,\delta}$, dependent on the desired confidence value δ (details in Appendix C.7.2). This is a standard assumption in literature [194]. Now, we state our main results below.

Theorem 5.2.1. *For large enough β, we have $\mathbb{E}_{\boldsymbol{s}\sim\mu_0,\boldsymbol{a}\sim\pi(\cdot|\boldsymbol{s})}[\hat{Q}^{\pi}(\boldsymbol{s}, \boldsymbol{a})] \leq \mathbb{E}_{\boldsymbol{s}\sim\mu_0,\boldsymbol{a}\sim\pi(\cdot|\boldsymbol{s})}[Q^{\pi}(\boldsymbol{s}, \boldsymbol{a})]$, where $\mu_0(\boldsymbol{s})$ is the initial state distribution. Furthermore, when ϵ_s is small, such as in the large sample regime, or when the model bias ϵ_m is small, a small β is sufficient to guarantee this condition along with an appropriate choice of f.*

The proof for Theorem 5.2.1 can be found in Appendix C.7.2. Finally, while Kumar et al. [181] also analyze how regularized value function training can provide lower bounds on the value function at each state in the dataset [181] (Proposition 3.1-3.2), *our result shows that COMBO is less conservative in that it does not underestimate the value function at every state in the dataset like CQL (Remark C.7.5)* and might even overestimate these values. Instead COMBO penalizes Q-values at states generated via model rollouts from $\rho(\boldsymbol{s}, \boldsymbol{a})$. While it is challenging to argue that that either COMBO or CQL attains the tightest possible lower-bound on return, in our final result of this section, we discuss a sufficient condition for the COMBO lower-bound to be tighter than CQL.

Theorem 5.2.2 (CQL vs COMBO). *Let* $\Delta^{\pi}_{COMBO} := \mathbb{E}_{\boldsymbol{s},\boldsymbol{a}\sim d_{\overline{\mathcal{M}}}(\boldsymbol{s}),\pi(\boldsymbol{a}|\boldsymbol{s})}\left[\hat{Q}^{\pi}(\boldsymbol{s},\boldsymbol{a})\right]$ *and* $\Delta^{\pi}_{CQL} := \mathbb{E}_{\boldsymbol{s},\boldsymbol{a}\sim d_{\overline{\mathcal{M}}}(\boldsymbol{s}),\pi(\boldsymbol{a}|\boldsymbol{s})}\left[\hat{Q}^{\pi}_{CQL}(\boldsymbol{s},\boldsymbol{a})\right]$ *denote the average values on the dataset under the Q-functions learned by COMBO and CQL respectively. Then,* $\Delta^{\pi}_{COMBO} \geq \Delta^{\pi}_{CQL}$, *if:*

$$\mathbb{E}_{\boldsymbol{s},\boldsymbol{a}\sim\rho(\boldsymbol{s},\boldsymbol{a})}\left[\frac{\pi(\boldsymbol{a}|\boldsymbol{s})}{\pi_{\beta}(\boldsymbol{a}|\boldsymbol{s})}\right] - \mathbb{E}_{\boldsymbol{s},\boldsymbol{a}\sim d_{\overline{\mathcal{M}}}(\boldsymbol{s}),\pi(\boldsymbol{a}|\boldsymbol{s})}\left[\frac{\pi(\boldsymbol{a}|\boldsymbol{s})}{\pi_{\beta}(\boldsymbol{a}|\boldsymbol{s})}\right] \leq 0. \qquad (*)$$

Proposition 5.2.2 indicates that COMBO will be less conservative than CQL when the action probabilities under learned policy $\pi(\boldsymbol{a}|\boldsymbol{s})$ and the probabilities under the behavior policy $\pi_{\beta}(\boldsymbol{a}|\boldsymbol{s})$ are closer together on state-action tuples drawn from $\rho(\boldsymbol{s},\boldsymbol{a})$ (i.e., sampled from the model using the policy $\pi(\boldsymbol{a}|\boldsymbol{s})$), than they are on states from the dataset and actions from the policy, $d_{\overline{\mathcal{M}}}(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s})$. COMBO's objective (Equation 5.2.1) only penalizes Q-values under $\rho(\boldsymbol{s},\boldsymbol{a})$, which, in practice, are expected to primarily consist of out-of-distribution states generated from model rollouts, and does not penalize the Q-value at states drawn from $d_{\overline{\mathcal{M}}}(\boldsymbol{s})$. As a result, the expression $(*)$ is likely to be negative, making COMBO less conservative than CQL.

5.2.4.2 *Model-Based Safe Policy Improvement*

Now that we have shown various aspects of the lower-bound on the Q-function induced by COMBO, we provide policy improvement guarantees for the COMBO algorithm. As mentioned earlier, we consider a simplified variant for COMBO for this result, which utilizes a Bellman backup operator that interpolates between the model-based backup operator and model-free backup operator, with a weight equal to the fraction f. Formally, Proposition 5.2.3 discuss safe improvement guarantees over the behavior policy, building on prior work [193, 181].

Theorem 5.2.3 (ζ-safe policy improvement). *Let* $\hat{\pi}_{out}(\boldsymbol{a}|\boldsymbol{s})$ *be the policy obtained by COMBO. Then, the policy* $\hat{\pi}_{out}(\boldsymbol{a}|\boldsymbol{s})$ *is a* ζ*-safe policy improvement over* π_{β} *in the actual MDP* $\mathcal{M}$, *i.e.,* $J(\hat{\pi}_{out},\mathcal{M}) \geq J(\pi_{\beta},\mathcal{M}) - \zeta$, *with probability at least* $1-\delta$, *where* ζ *is given by,*

$$\mathcal{O}\left(\frac{\gamma f}{(1-\gamma)^2}\right) \underbrace{\mathbb{E}_{\boldsymbol{s}\sim d^{\hat{\pi}_{out}}_{\mathcal{M}}}\left[\sqrt{\frac{|\mathcal{A}|}{|\mathcal{D}(\boldsymbol{s})|}}\mathrm{D}_{CQL}(\hat{\pi}_{out},\pi_{\beta})\right]}_{:=\,(1)}$$
$$+\mathcal{O}\left(\frac{\gamma(1-f)}{(1-\gamma)^2}\right)\underbrace{\mathrm{D}_{\mathrm{TV}}(\overline{\mathcal{M}},\widehat{\mathcal{M}})}_{:=\,(2)} - \underbrace{\beta\frac{\nu(\rho^{\pi},f)-\nu(\rho^{\beta},f)}{(1-\gamma)}}_{:=\,(3)}.$$

The complete statement (with constants and terms that grow smaller than quadratic in the horizon) and proof for Proposition 5.2.3 is provided in Appendix C.7.4. D_{CQL} denotes a notion of probabilistic distance between policies [181] which we discuss further in Appendix C.7.4. The expression for ζ in Proposition 5.2.3 consists of three terms: term (1) captures the decrease in the policy performance due to limited data, and decays as the size of $\mathcal{D}$ increases. The second term (2) captures the suboptimality induced by the bias in the learned model. Finally, as we show in Appendix C.7.4, the third term (3) is equivalent to the improvement in policy performance as a result of running COMBO in the empirical and model MDPs. Since the learned model is trained on the dataset $\mathcal{D}$ with transitions generated from the behavior policy π_β, the marginal distribution $\rho^\beta(\boldsymbol{s}, \boldsymbol{a})$ is expected to be closer to $d(\boldsymbol{s}, \boldsymbol{a})$ for π_β as compared to the counterpart for the learned policy, ρ^π. Thus, term (3) is expected to be positive in practice, and in such cases, an appropriate (large) choice of β will make term (3) large enough to counteract terms (1) and (2) that reduce policy performance. We discuss this elaborately in Appendix C.7.4 (Remark C.7.8).

Further note that in contrast to Proposition 3.6 in Kumar et al. [181], note that our result indicates the sampling error (term (1)) is reduced (multiplied by a fraction f) when a near-accurate model is used to augment data for training the Q-function, and similarity, it can avoid the bias of model-based methods by relying more on the model-free component. This allows COMBO to attain the best-of-both model-free and model-based methods, via a suitable choice of the fraction f. To summarize, through an appropriate choice of f, Proposition 5.2.3 guarantees safe improvement over the behavior policy without requiring access to an oracle uncertainty estimation algorithm.

5.2.5 *Experimental Evaluation of COMBO*

In our experiments, we aim to answer the follow questions: **(1)** Can COMBO generalize better than previous offline model-free and model-based approaches in a setting that requires generalization to tasks that are different from what the behavior policy solves? **(2)** How does COMBO compare with prior work in tasks with high-dimensional image observations?, **(3)** How does COMBO compare to prior offline model-free and model-based methods in standard offline RL benchmarks?

To answer those questions, we compare COMBO to several prior methods. In the domains with compact state spaces, we compare with recent model-free algorithms like BEAR [179], BRAC [343], and the CQL [181] algorithm presented in the previous section; as well as MOPO [365] and MOReL [166] which are two recent model-based algorithms. In addition, we also compare with an offline version of SAC [126] (denoted as SAC-off), and behavioral cloning (BC). In high-dimensional image-based domains, which we use to answer question (3), we compare to LOMPO [272], which is a latent space offline model-based RL method

Environment	BatchMean	BatchMax	COMBO (Ours)	MOPO	MOReL	CQL
halfcheetah-jump	-1022.6	1808.6	**5392.7**	4016.6	3228.7	741.1
ant-angle	866.7	2311.9	**2764.8**	2530.9	2660.3	2473.4
sawyer-door-close	5%	100%	**100**%	65.8%	42.9%	36.7%

Table 5: Average returns of halfcheetah-jump and ant-angle and average success rate of sawyer-door-close that require out-of-distribution generalization. All results are averaged over 3 random seeds. We include the mean and max undiscounted return / success rate of the episodes in the batch data (under Batch Mean and Batch Max, respectively) for comparison.

that handles image inputs, latent space MBPO (denoted LMBPO), similar to Janner et al. [147] which uses the model to generate additional synthetic data, the fully offline version of SLAC [196] (denoted SLAC-off), which only uses a variational model for state representation purposes, and CQL from image inputs. To our knowledge, CQL, MOPO, and LOMPO are representative of state-of-the-art model-free and model-based offline RL methods. Hence we choose them as comparisons to COMBO. For more details of our experimental set-up, comparisons, and hyperparameters, see Appendix C.8.

5.2.5.1 *Results on tasks that require generalization*

To answer question (1), we use the two environments halfcheetah-jump and ant-angle constructed in Yu et al. [365], which requires the agent to solve a task that is different from what the behavior policy solved. In both environments, the offline dataset is collected by policies trained with the original reward functions of halfcheetah and ant, which reward the halfcheetah and the ant to run as fast as possible. The behavior policies are trained with SAC with 1M steps and we take the full replay buffer as the offline dataset. Following Yu et al. [365], we relabel the rewards in the offline datasets to reward the halfcheetah to jump as high as possible and the ant to run to the top corner with a 30 degree angle as fast as possible. Following the same manner, we construct a third task sawyer-door-close based on the environment in Yu et al. [364], Rafailov et al. [272]. In this task, we collect the offline data with SAC policies trained with a sparse reward function that only gives a reward of 1 when the door is *opened* by the sawyer robot and 0 otherwise. The offline dataset is similar to the "medium-expert" dataset in the D4RL benchmark since we mix equal amounts of data collected by a fully-trained SAC policy and a partially-trained SAC policy. We relabel the reward such that it is 1 when the door is *closed* and 0 otherwise. Therefore, in these datasets, the offline RL methods must generalize beyond behaviors in the offline data in order to learn the intended behaviors. We visualize sawyer-door-close on the right in Figure 20 in Appendix C.8.5.

We present the results on the three tasks in Table 5. COMBO significantly outperforms MOPO, MOReL and CQL, two representative model-based methods and one representative model-free methods respectively, in the halfcheetah-jump and sawyer-door-close tasks, and achieves an approximately 8%, 4% and 12% improvement over MOPO, MOReL and

Dataset	Environment	COMBO (Ours)	LOMPO	LMBPO	SLAC-Off	CQL
M-R	walker_walk	**69.2**	66.9	59.8	45.1	15.6
M	walker_walk	57.7	60.2	**61.7**	41.5	38.9
M-E	walker_walk	76.4	**78.9**	47.3	34.9	36.3
expert	walker_walk	**61.1**	55.6	13.2	12.6	43.3
M-E	sawyer-door	**100.0%**	**100.0%**	0.0%	0.0%	0.0%
expert	sawyer-door	**96.7%**	0.0%	0.0%	0.0%	0.0%

Table 6: Results for vision experiments. For the Walker task each number is the normalized score proposed in [92] of the policy at the last iteration of training, averaged over 3 random seeds. For the Sawyer task, we report success rates over the last 100 evaluation runs of training. For the dataset, M refers to medium, M-R refers to medium-replay, and M-E refers to medium expert.

CQL respectively on the ant-angle task. These results validate that COMBO achieves better generalization results in practice by behaving less conservatively than prior model-free offline methods (compare to CQL, which doesn't improve much) and more robustly than prior model-based offline methods (compare to MOReL and MOPO).

5.2.5.2 *Results on Image-Based Tasks*

To answer question (2), we evaluate COMBO on two image-based tasks: the standard walker (walker-walk) task from the the DM Control suite [319] and a visual door opening environment with a Sawyer robotic arm (sawyer-door) as used in Section 5.2.5.1. For the walker task we construct 4 datasets: medium-replay (M-R), medium (M), medium-expert (M-E), and expert, similar to Fu et al. [92], each consisting of 200 trajectories. For sawyer-door task we use only the medium-expert and the expert datasets, due to the sparse reward – the agent is rewarded only when it successfully opens the door. Both environments are visulized in Figure 20 in Appendix C.8.5. To extend COMBO to the image-based setting, we follow Rafailov et al. [272] and train a recurrent variational model using the offline data and use train COMBO in the latent space of this model.

We present results in Table 6. On the walker-walk task, COMBO performs in line with LOMPO and previous methods. On the more challenging Sawyer task, COMBO matches LOMPO and achieves 100% success rate on the medium-expert dataset, and substantially outperforms all other methods on the narrow expert dataset, achieving an average success rate of 96.7%, when all other model-based and model-free methods fail.

5.2.5.3 *Results on the D4RL Tasks*

Finally, to answer the question (3), we evaluate COMBO on the OpenAI Gym [32] domains in the D4RL benchmark [92][2], which contains three environments (halfcheetah, hopper, and walker2d) and four dataset types (random, medium, medium-replay, and medium-expert). We include the results in Table 7. The numbers of BC, SAC-off, BEAR, BRAC-P and BRAC-v are taken from the D4RL paper, while the results for MOPO, MOReL

[2] Note that these results are on the older "v0" versions of the D4RL tasks, which are different from "v2" versions. Methods attain significantly different performance numbers on v2 compared to v0.

Dataset type	Environment	BC	COMBO (ours)	MOPO	MOReL	CQL	SAC-off	BEAR	BRAC-p	BRAC-v
random	halfcheetah	2.1	**38.8**	35.4	25.6	35.4	30.5	25.1	24.1	31.2
random	hopper	1.6	17.9	11.7	**53.6**	10.8	11.3	11.4	11.0	12.2
random	walker2d	9.8	7.0	13.6	**37.3**	7.0	4.1	7.3	-0.2	1.9
medium	halfcheetah	36.1	**54.2**	42.3	42.1	44.4	-4.3	41.7	43.8	46.3
medium	hopper	29.0	**97.2**	28.0	95.4	86.6	0.8	52.1	32.7	31.1
medium	walker2d	6.6	**81.9**	17.8	77.8	74.5	0.9	59.1	77.5	81.1
medium-replay	halfcheetah	38.4	**55.1**	53.1	40.2	46.2	-2.4	38.6	45.4	47.7
medium-replay	hopper	11.8	89.5	67.5	**93.6**	48.6	3.5	33.7	0.6	0.6
medium-replay	walker2d	11.3	**56.0**	39.0	49.8	32.6	1.9	19.2	-0.3	0.9
med-expert	halfcheetah	35.8	**90.0**	63.3	53.3	62.4	1.8	53.4	44.2	41.9
med-expert	hopper	111.9	**111.1**	23.7	108.7	111.0	1.6	96.3	1.9	0.8
med-expert	walker2d	6.4	**103.3**	44.6	95.6	98.7	-0.1	40.1	76.9	81.6

Table 7: Results on v0-D4RL datasets. Each number is the normalized score proposed in [92] of the policy at the last iteration of training, averaged over 3 random seeds. We take the results of MOPO, MOReL and CQL from their original papers and results of other model-free methods from [92]. We include the performance of behavior cloning (**BC**) from offline datasets for comparison. We bold the highest score across all methods.

and CQL are based on their respective papers [365, 181]. COMBO achieves the best performance in 9 out of 12 settings and comparable result in 1 out of the remaining 3 settings (hopper medium-replay). As noted by Yu et al. [365] and Rafailov et al. [272], model-based offline methods are generally more performant on datasets that are collected by a wide range of policies and have diverse state-action distributions (random, medium-replay datasets) while model-free approaches do better on datasets with narrow distributions (medium, medium-expert datasets). However, in these results, COMBO generally performs well across dataset types compared to existing model-free and model-based approaches, suggesting that COMBO is robust to different datasets.

5.3 DISCUSSION AND LIMITATIONS

In this chapter, we presented an algorithmic framework for estimating value functions in offline RL that lower bound the true policy value function. Optimizing the policy against such as a lower bound value function results in policies that provably improve over the behavioral policy, from the perspective of robust or safe policy improvement. Learning this sort of a conservative value function removes the need to use policy constraints, thereby alleviating the need to estimate the support of the behavioral policy in high-dimensional action spaces. Empirically, we demonstrate that our practical approaches, CQL and COMBO outperform prior offline RL methods on a wide range of offline RL benchmark tasks, including complex control tasks and tasks with raw image observations. Externally of work covered in this thesis, CQL has been utilized in several real-world applications: **(1)** for optimizing decisions in notification systems [267], and **(2)** for identifying dead end in patient treatment in healthcare [167].

Although conservative approaches have been applied in various downstream applications, there are still several unresolved questions that need to be addressed. Firstly, like any machine learning method, the performance of offline RL methods heavily relies on the values assigned to specific hyperparameters. These include the choice of the model architecture used to represent the Q-function and, more importantly, the hyperparameter α in CQL (and β in COMBO), which governs the strength of the conservatism regularizer. In general, finding a universal recipe for tuning these hyperparameters is a challenging task. However, it is possible to develop strategies to select these hyperparameters under certain assumptions and conditions specific to the underlying offline RL problem. For instance, our follow-up work presents an empirical approach to tuning this hyperparameter in the context of robotic applications [184].

From a theoretical standpoint, while we show that conservative methods learn lower bounds on the Q-function in tabular, linear, and a subset of non-linear function approximation cases, a rigorous theoretical analysis of CQL with deep neural nets is yet to be conducted. As we discuss in Chapter 6, the implicit regularization effects of non-linear neural net models may interact poorly with value-based RL, and understanding its consequences for conservative methods remains an open question.

ACKNOWLEDGEMENTS AND FUNDING

We thank Mohammad Norouzi, Oleh Rybkin, Anton Raichuk, Avi Singh, Vitchyr Pong and anonymous reviewers from the Robotic AI and Learning Lab at UC Berkeley and NeurIPS for their feedback on an earlier version of this paper. We thank Rishabh Agarwal for help with the Atari QR-DQN/REM codebase and for sharing baseline results. This research was funded by the DARPA Assured Autonomy program, and compute support from Google and Amazon.

Part III

EXTENSIONS OF OFFLINE RL

6

OFFLINE RL WITH LARGE MODELS

Abstract

So far, we have developed algorithmic approaches for offline RL. In practice, we would often want to instantiate these algorithms in a way that allows them to convert large amounts of data into performant control policies. One way to do so is by training high-capacity neural network value functions and policies with diverse offline datasets. In support is the observation that this kind of recipe is quite effective in supervised learning: despite the massive over-parameterization, deep networks trained via supervised learning are easy to optimize and exhibit excellent generalization. One hypothesis to explain this is that overparameterized deep networks enjoy the benefits of implicit regularization induced by stochastic gradient descent (SGD), which favors parsimonious solutions that generalize well on test inputs. It is reasonable to surmise that deep reinforcement learning (RL) methods could also benefit from this effect. In this chapter, on the contrary, we illustrate that the implicit regularization effect of SGD seen in supervised learning could, in fact, be harmful in the offline deep RL setting, leading to poor generalization and degenerate feature representations. Our theoretical analysis shows that when existing models of implicit regularization are applied to temporal difference learning (TD-learning), the resulting derived regularizer favors degenerate solutions with excessive "aliasing". We back up these findings empirically and propose a simple and effective explicit regularizer, called DR3, that counteracts this undesirable effect. We then extend DR3 to a recipe that is able to successfully train 80 million parameter ResNet models entirely via offline TD-learning to outperform variants of imitation with heterogeneous data.

6.1 INTRODUCTION

Despite being massively over-parameterized, deep neural networks with billions of parameters still learn representations that generalize well when trained with supervised

learning. This is surprising considering the classical notions of overfitting when provided with many parameters. A widely-held consensus is that deep networks find simple solutions that generalize due to various implicit regularization effects [31, 341, 17, 117, 338, 210]. Does this mean that large, over-parameterized networks will also perform well for offline RL, scaling effectively as the number of parameters is increased?

In the first part of this chapter (Section 6.2), we show that, the very same implicit regularization effects can lead to the emergence of poor representations when training overparameterized deep network value functions via offline RL. This often results in the inability to improve performance with larger models and, in particular, unreliable optimization behavior over the course of offline RL training. While there are already some empirical studies showing that value function training via offline temporal-difference (TD) learning leads to emergence of poor representations [182], our goal in this chapter is to understand the cause in a simpler theoretical model and develop a potential solution. Building on the theoretical framework developed by Blanc et al. [31], Damian et al. [52], we characterize the implicit regularizer that arises when training deep value functions with offline TD learning. The form of this implicit regularizer implies that offline TD-learning would "co-adapt" the learned feature representations at state-action tuples that appear on either side of a Bellman backup.

Practically, we illustrate that this theoretically-predicted co-adaptation phenomenon results in a higher dot product of the features of consecutive state-action tuples learned by the Q-value network (Section 6.2.1). Training runs that exhibit feature co-adaptation typically converge to poorly performing solutions. Even when Q-values are not overestimated, prolonged training in offline RL can result in performance degradation as feature co-adaptation increases. To mitigate this co-adaptation issue, which arises as a result of implicit regularization, we propose an *explicit regularizer* that we call DR3 (Section 6.2.3). While exactly estimating the effects of the theoretically derived implicit regularizer is computationally difficult, DR3 provides a simple and tractable approximation that mitigates the issues discussed above. In practice, DR3 amounts to regularizing the features at consecutive state-action pairs to be dissimilar in terms of their dot-product similarity. Empirically, we find that DR3 prevents previously noted pathologies such as feature rank collapse [182], gives methods that train for longer and improves performance relative to the base offline RL method on benchmark problems.

Building on this approach, in the second part of this chapter (Section 6.3), we perform a large-scale empirical study that aims to train a large model via offline conservative Q-learning. Specifically, we train a single policy to play 40 Atari games [22], similar to Lee et al. [199], and evaluate performance when the training dataset contains expert trajectories *and* when the data is sub-optimal. This problem is especially challenging due to the diversity in dynamics, rewards, visuals, and agent embodiments across different games.

Furthermore, the sub-optimal data setting requires the learning algorithm to "stitch together" useful segments of sub-optimal trajectories to perform well. Our approach, **Scaled Q-learning** (Scaled QL), incorporates a variant of the DR3 technique, which normalizes features instead of using regularization to attain the same benefit without needing a hyperparameter, in conjunction with carefully chosen neural network architectural design decisions. Scaled QL can train up to 80 million parameter ResNet [131] models entirely via offline RL, and the performance of our approach follows a power-law relationship between model capacity and performance, similar to various scaling studies in supervised and unsupervised learning. In terms of absolute performance, scaled Q-learning learns policies that attain more than 100% human-level performance on most of the 40 games, about **2x** better than prior supervised learning (SL) approaches for learning from sub-optimal offline data (51% human-level performance).

6.2 DR3: EXPLICIT REGULARIZATION FOR VALUE-BASED OFFLINE RL

While the "deadly-triad" [314] suggests that training value function approximators with bootstrapping off-policy can lead to divergence, modern deep RL algorithms have been able to successfully combine these properties [328]. However, making too many TD updates to the Q-function in offline deep RL is known to sometimes lead to performance degradation and unlearning, even for otherwise effective modern algorithms [90, 82, 6, 182]. Such unlearning is not typically observed when training overparameterized models via supervised learning, so what about TD learning is responsible for it? We show that one possible explanation behind this pathology is the implicit regularization induced by minimizing TD error on a deep Q-network. Our theoretical results suggest that this implicit regularization "co-adapts" the representations of state-action pairs that appear in a Bellman backup (we will define this more precisely below). Empirically, this typically manifests as "co-adapted" features for consecutive state-action tuples, even with specialized TD-learning algorithms that account for distributional shift, and this in turn leads to poor final performance both in theory and in practice. We first provide empirical evidence of this co-adaptation phenomenon in Section 6.2.1 (additional evidence in Appendix D.1.1) and then theoretically characterize the implicit regularization in TD learning, and discuss how it can explain the co-adaptation phenomenon in Section 6.2.2.

6.2.1 *Feature Co-Adaptation And Implicit Regularization*

In this section, we empirically identify a *feature co-adaptation* phenomenon that appears when training value functions via bootstrapping, where the feature representations of consecutive state-action pairs exhibit a large value of the dot product $\phi(\boldsymbol{s}, \boldsymbol{a})^\top \phi(\boldsymbol{s}', \boldsymbol{a}')$. Note that feature co-adaptation may arise because of high cosine similarity or because

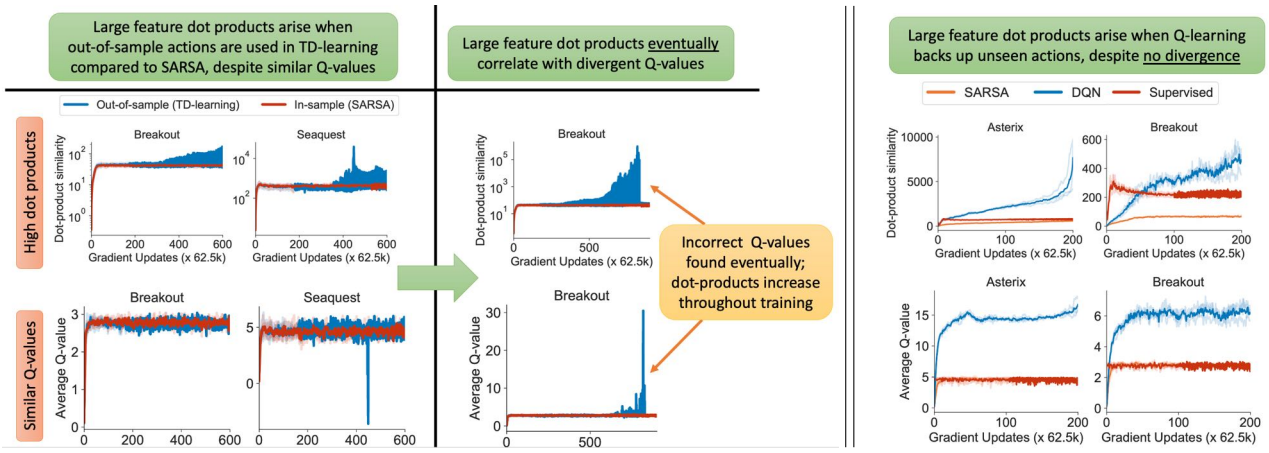


Figure 13: Feature dot-products $\phi(\boldsymbol{s}, \boldsymbol{a})^\top \phi(\boldsymbol{s}', \boldsymbol{a}')$ increase during training when backing up from *out-of-sample* but in-distribution actions (**TD-learning**: left, **Q-learning**: right), though the average Q-value converges and stays relatively constant. Using only seen state-action pairs for backups (**offline SARSA**) or not performing Bellman backups (i.e., **supervised regression**) avoids this issue, with stable and relatively low dot products. *Left*: TD-learning with high feature dot products eventually destabilizes and produces incorrect Q-values, *Right*: DQN attains extremely large feature dot products, despite a relatively stable trend in Q-values.

of high feature norms. Feature co-adaptation appears even when there is no explicit objective to increase feature similarity.

Experimental setup. We ran supervised regression and three variants of approximate dynamic programming (ADP) on an offline dataset consisting of 1% of uniformly-sampled data from the replay buffer of DQN on two Atari games, previously used in Agarwal et al. [6]. First, for comparison, we trained a Q-function via **supervised regression** to Monte-Carlo (MC) return estimates on the offline dataset to estimate the value of the behavior policy. Then, we trained variants of ADP which differ in the selection procedure for the action $\boldsymbol{a}'$ that appears in the target value in the TD-error. The **offline SARSA** variant aims to estimate the value of the behavior policy, Q^{π_β}, and sets $\boldsymbol{a}'$ to the actual action observed at the next time step in the dataset, such that $(\boldsymbol{s}', \boldsymbol{a}') \in \mathcal{D}$. The **TD-learning** variant also aims to estimate the value of the behavior policy, but utilizes the expectation of the target Q-value over actions $\boldsymbol{a}'$ sampled from the behavior policy π_β, $\boldsymbol{a}' \sim \pi_\beta(\cdot|\boldsymbol{s}')$. We do not have access to the functional form of π_β for the experiment shown in Figure 13 since the dataset corresponds to the behavior policy induced by the replay buffer of an online DQN, so we train a model for this policy using supervised learning. However, we see similar results comparing **offline SARSA** and **TD-learning** on a gridworld domain where we can access the exact functional form of the behavior policy in Appendix D.1.5.2. All of the methods so far estimate Q^{π_β} using different target value estimators. We also train **Q-learning**, which chooses the action $\boldsymbol{a}'$ to maximize the learned Q-function. While

Q-learning learns a different Q-function, we can still compare the relative stability of these methods to gain intuition about the learning dynamics. In addition to feature dot products $\phi(\boldsymbol{s}, \boldsymbol{a})^\top \phi(\boldsymbol{s}', \boldsymbol{a}')$, we also track the average prediction of the Q-network over the dataset to measure whether the predictions diverge or are stable in expectation.

Observing feature co-adaptation empirically. As shown in Figure 13 (right), the average dot product (top row) between features at consecutive state-action tuples continuously increases for both Q-learning and TD-learning (after enough gradient steps), whereas it flatlines and converges to a small value for supervised regression. We might at first think that this is simply a case of Q-learning failing to converge. However, the bottom row shows that the average Q-values do in fact converge to a stable value. Despite this, the optimizer drives the network towards higher feature dot products. There is no explicit term in the TD error objective that encourages this behavior, indicating the presence of some implicit regularization phenomenon. This *implicit* preference towards maximizing the dot products of features at consecutive state-action tuples is what we call "feature co-adaptation."

When does feature co-adaptation emerge? Observe in Figure 13 (right) that the feature dot products for offline SARSA converge quickly and are relatively flat, similarly to supervised regression. This indicates that utilizing a bootstrapped update alone is not responsible for the increasing dot-products and instability, because while offline SARSA uses backups, it behaves similarly to supervised MC regression. Unlike offline SARSA, feature co-adaptation emerges for TD-learning, which is surprising as TD-learning also aims to estimate the value of the behavior policy, and hence should match offline SARSA in expectation. The key difference is that while offline SARSA always utilizes actions $\boldsymbol{a}'$ observed in the training dataset for the backup, TD-learning may utilize potentially unseen actions $\boldsymbol{a}'$ in the backup, even though these actions $\boldsymbol{a}' \sim \pi_\beta(\cdot|\boldsymbol{s}')$ are *within* the distribution of the data-generating policy. This suggests that utilizing **out-of-sample** actions in the Bellman backup, even when they are not out-of-distribution, critically alters the learning dynamics. This is distinct from the more common observation in offline RL, which attributes training challenges to out-of-distribution actions, but not out-of-sample actions. The theoretical model developed in Section 6.2.2 will provide an explanation for this observation with a discussion about how feature co-adaption caused due to out-of-sample actions can be detrimental in offline RL.

6.2.2 *Theoretically Characterizing Implicit Regularization in TD-Learning*

Why does feature co-adaptation emerge in TD-learning and what do *out-of-sample* actions have to do with it? To answer this question, we theoretically characterize the implicit regularization effects in TD-learning. We analyze the learning dynamics of TD learning in the overparameterized regime, where there are many different parameter vectors θ

that fully minimize the training set temporal difference error. We base our analysis of TD learning on the analysis of implicit regularization in supervised learning, previously developed by Blanc et al. [31], Damian et al. [52].

Background. When training an overparameterized $f_\theta(\boldsymbol{x})$ via supervised regression using the squared loss, denoted by L, many different values of θ will satisfy $L(\theta) = 0$ on the training set due to overparameterization, but Blanc et al. [31] show that the dynamics of stochastic gradient descent will only find fixed points θ^* that additionally satisfy a condition which can be expressed as $\nabla_\theta R(\theta^*) = 0$, along certain directions (that we will describe shortly). This function $R(\theta)$ is referred to as the implicit regularizer. The noisy gradient updates analyzed in this model have the form:

$$\theta_{k+1} \leftarrow \theta_k - \eta \nabla_\theta L(\theta) + \eta \varepsilon_k, \quad \varepsilon_k \sim \mathcal{N}(0, M). \tag{6.2.1}$$

Blanc et al. [31] and Damian et al. [52] show that some common SGD techniques fall into this framework, for example, when the regression targets in supervised learning are corrupted with $\mathcal{N}(0,1)$ label noise, then the resulting $M = \sum_{i=1}^{|\mathcal{D}|} \nabla_\theta f_\theta(\boldsymbol{x}_i) \nabla_\theta f_\theta(\boldsymbol{x}_i)^\top$ and the induced implicit regularizer R is given by $R(\theta) = \sum_i^{|\mathcal{D}|} ||\nabla_\theta f_\theta(\boldsymbol{x}_i)||_2^2$. Any solution θ^* found by Equation 6.2.1 must satisfy $\nabla_\theta R(\theta^*) = 0$ along directions $\boldsymbol{v} \in \mathbb{R}^{|\theta|}$ which lie in the null space of the Hessian of the loss $\nabla_\theta^2 L(\theta^*)$ at θ^*, $\boldsymbol{v} \in \text{Null}(\nabla_\theta^2 L(\theta^*))$. The intuition behind the implicit regularization effect is that along such directions in the parameter space, the Hessian is unable to contract θ_k when running noisy gradient updates (Equation 6.2.1). Therefore, the only condition that the noisy gradient updates converge/stabilize at θ^* is given by the condition that $\nabla R(\theta^*) = 0$. This model corroborates findings [233, 52] about the solutions from SGD, which motivates our use.

Our setup. Following this framework, we analyze the fixed points of noisy TD-learning. We consider noisy pseudo-gradient (or semi-gradient) TD updates with a general noise covariance M:

$$\theta_{k+1} = \theta_k - \eta \underbrace{\left(\sum_i \nabla_\theta Q(\boldsymbol{s}_i, \boldsymbol{a}_i) \left(Q_\theta(\boldsymbol{s}_i, \boldsymbol{a}_i) - (r_i + \gamma Q_\theta(\boldsymbol{s}_i', \boldsymbol{a}_i')) \right) \right)}_{:= g(\theta)} + \eta \varepsilon_k, \quad \varepsilon_k \sim \mathcal{N}(0, M) \tag{6.2.2}$$

We use a deterministic policy $\boldsymbol{a}_i' = \pi(\boldsymbol{s}_i')$ to simplify exposition. Following Damian et al. [52], we can set the noise model M as $M = \sum_i \nabla_\theta Q(\boldsymbol{s}_i, \boldsymbol{a}_i) \nabla_\theta Q(\boldsymbol{s}_i, \boldsymbol{a}_i)^\top$, or utilize a different choice of M, but we will derive the general form first. Let θ^* denote a stationary point of the training TD error, such that the pseudo-gradient $g(\theta^*) = 0$. Further, we denote the derivative of $g(\theta)$ w.r.t. θ as the matrix $G(\theta) \in \mathbb{R}^{|\theta| \times |\theta|}$, and refer to it as the *pseudo-Hessian*: although $G(\theta)$ is not actually the second derivative of any well-defined objective, since TD updates are not proper gradient updates, as we will see it will play a

similar role to the Hessian in gradient descent. For brevity, define $G = G(\theta^*)$, $g = g(\theta^*)$, $\nabla G = \nabla_\theta G(\theta^*) \in \mathbb{R}^{|\theta| \times |\theta| \times |\theta|}$, and let $\lambda_i(P)$ denote the i-th eigenvalue of matrix P, when arranged in decreasing order of its (complex) magnitude $|\lambda_i(P)|$ (note that eigenvalues can be complex for non-symmetric matrices that we encounter here).

Assumptions. To simplify analysis, we assume that matrices G and M (i.e., the noise covariance matrix) span the same n-dimensional basis in d-dimensional space, where d is the number of parameters and n is the number of datapoints, and $n \ll d$ due to overparameterization. We also require θ^* to satisfy a technical criterion that requires approximate alignment between the eigenspaces of G and the gradient of the Q-function, without which noisy TD may not be stable at θ^*. We summarize all the assumptions in Appendix D.3, and present the resulting regularizer below.

Theorem 6.2.1 (Implicit regularizer at TD fixed points)**.** *Under the assumptions so far, a fixed point of TD-learning, θ^*, where $Q_{\theta^*}(\boldsymbol{s}_i, \boldsymbol{a}_i) = r_i + \gamma Q_{\theta^*}(\boldsymbol{s}'_i, \boldsymbol{a}'_i)$ for every $(\boldsymbol{s}_i, \boldsymbol{a}_i, \boldsymbol{s}'_i) \in \mathcal{D}$ is stable (atttractive) if:* ***(1)*** *it satisfies $\mathrm{Re}(\lambda_i(G)) \geq 0, \forall i$ and $\mathrm{Re}(\lambda_i(G)) > 0$ if $|\mathrm{Imag}(\lambda_i(G))| > 0$, and* ***(2)*** *along directions $\boldsymbol{v} \in \mathbb{R}^{dim(\theta)}, \boldsymbol{v} \in Null(G)$, θ^* is the stationary point of the implicit regularizer:*

$$R_{\mathrm{TD}}(\theta) = \underbrace{\eta \sum_{i=1}^{|\mathcal{D}|} \nabla Q_\theta(\boldsymbol{s}_i, \boldsymbol{a}_i)^\top \Sigma^*_M \nabla Q_\theta(\boldsymbol{s}_i, \boldsymbol{a}_i)}_{\text{implicit regularizer for noisy GD}} \underbrace{- \eta\gamma \sum_{i=1}^{|\mathcal{D}|} \mathrm{tr}\left(\left[\left[\nabla Q_\theta(\boldsymbol{s}'_i, \boldsymbol{a}'_i)^\top \right]\right]^\top \Sigma^*_M \nabla Q_\theta(\boldsymbol{s}_i, \boldsymbol{a}_i) \right)}_{\text{additional term in TD learning}} \tag{6.2.3}$$

where $(\boldsymbol{s}_i, \boldsymbol{a}_i)$ and $(\boldsymbol{s}'_i, \boldsymbol{a}'_i)$ denote state-action pairs that appear together in a Bellman update, $[[\square]]$ denotes the stop-gradient function, which does not pass partial gradients w.r.t. θ into $\square$. Σ^*_M is the fixed point of the discrete Lyapunov equation:

$$\Sigma^*_M := (I - \eta G)\Sigma^*_M (I - \eta G)^\top + \eta^2 M.$$

A proof of Theorem 6.2.1 is provided in Appendix D.3. Next, we explain the intuition behind this result and provide a proof sketch. To derive the induced implicit regularizer for a stable fixed point θ^* of TD error, we study the learning dynamics of noisy TD learning (Equation 6.2.2) initialized at θ^*, and derive conditions under which this noisy update would stay close to θ^* with multiple updates. This gives rise to the two conditions shown in Theorem 6.2.1 which can be understood as controlling stability in mutually exclusive directions in the parameter space. If condition **(1)** is not satisfied, then even under-parameterized TD will diverge away from θ^*, since $I - \eta G$ would be a non-contraction as the spectral radius, $\rho(I - \eta G) \geq 1$ in that case. Thus, $\theta_k - \theta^*$ will grow or not decrease in some direction. When **(1)** is satisfied for all directions in the parameter

space, there are still directions where both the real and imaginary parts of the eigenvalue $\lambda_i(G)$ are 0 due to overparameterization[1]. In such directions, learning is governed by the projection of the noise under the tensor ∇G, which appears in the Taylor expansion of $\theta_k - \theta^*$ around the point θ^*:

$$\theta_{k+1} = \theta_k - \eta \left(g + G(\theta_k - \theta^*) + \frac{1}{2}\nabla G[\theta_k - \theta^*, \theta_k - \theta^*] \right) + \varepsilon_k, \quad \varepsilon_k \sim \mathcal{N}(0, M) \tag{6.2.4}$$

$$\implies \nu_{k+1} = (I - \eta G)\nu_k - \frac{\eta}{2}\nabla G[\nu_k, \nu_k] + \varepsilon_k, \tag{6.2.5}$$

where we reparameterize in terms of $\nu_k := \theta_k - \theta^*$. The proof shows that θ^* is stable if it is a stationary point of the implicit regularizer R_{TD} (condition **(2)**), which ensures that total noise (i.e., accumulated ε_k over iterations k) accumulated by ∇G does not lead to a large deviation in ν_k in directions where $I - \eta G$ does not contract.

Interpretation of Theorem 6.2.1. While the choice of the noise model M will change the form of the implicit regularizer, in practice, the form of M is not known as this corresponds to the noise induced via SGD. We can consider choices of M for interpretation, but Theorem 6.2.1 is easy to qualitatively interpret for M such that $\Sigma^*_M = I$. In this case, we find that the implicit preference towards local minima of $R_{\text{TD}}(\theta)$ can explain feature co-adaptation. In this case, the regularizer is simpler:

$$R_{\text{TD}}(\theta) := \sum_i ||\nabla Q_\theta(\boldsymbol{s}_i, \boldsymbol{a}_i)||_2^2 - \gamma \nabla Q_\theta(\boldsymbol{s}_i, \boldsymbol{a}_i)^\top \nabla [[Q_\theta(\boldsymbol{s}'_i, \boldsymbol{a}'_i)]].$$

The first term is equal to the squared per-datapoint gradient norm, which is same as the implicit regularizer in supervised learning obtained by Blanc et al. [31], Damian et al. [52] with label noise. However, $R_{\text{TD}}(\theta)$ additionally includes a second term that is equal to the dot product of the gradient of the Q-function at the current and next states, $\nabla_\theta Q_\theta(\boldsymbol{s}_i, \boldsymbol{a}_i)^\top \nabla_\theta Q_\theta(\boldsymbol{s}'_i, \boldsymbol{a}'_i)$, and thus this term is effectively *maximized*. When restricted to the last-layer parameters of a neural network, this term is equal to the dot product of the features at consecutive state-action tuples: $\sum_i \nabla_\theta Q_\theta(\boldsymbol{s}_i, \boldsymbol{a}_i)^\top \nabla_\theta Q_\theta(\boldsymbol{s}'_i, \boldsymbol{a}'_i) = \sum_i \phi(\boldsymbol{s}_i, \boldsymbol{a}_i)^\top \phi(\boldsymbol{s}'_i, \boldsymbol{a}'_i)$. The tendency to maximize this quantity to attain a local minimizer of the implicit regularizer corroborates the empirical findings of increased dot product in Section 6.2.1.

Explaining the difference between utilizing seen and unseen actions in the backup. If all state-action pairs $(\boldsymbol{s}'_i, \boldsymbol{a}'_i)$ appearing on the right-hand-side of the Bellman update also appear in the dataset $\mathcal{D}$, as in the case of offline SARSA (Figure 13), the preference to increase dot products will be balanced by the affinity to reduce gradient norm (first term of $R_{\text{TD}}(\theta)$ when $\Sigma^*_M = I$): for example, for offline SARSA, when $(\boldsymbol{s}'_i, \boldsymbol{a}'_i)$ are permutations

[1] To see why this is the case, note that $\text{rank}(G) \leq |\mathcal{D}| \ll \dim(\theta)$, and so some eigenvalues of G are 0.

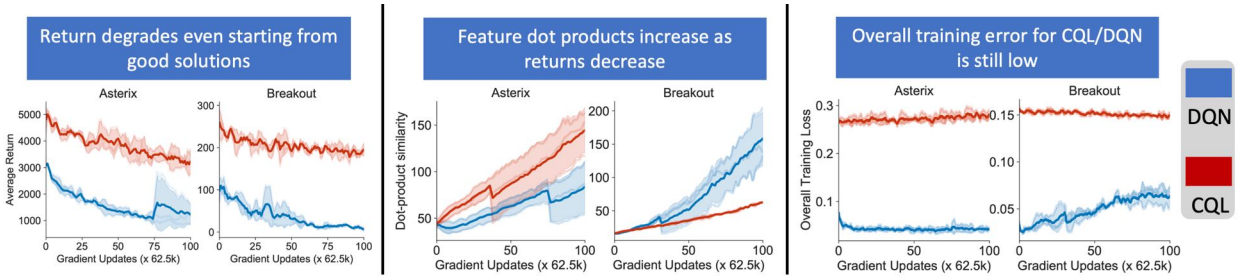


Figure 14: Even when current offline RL algorithms are initialized at a high-performing checkpoint that attains small feature dot products, feature dot products increase with further training and the performance degrades.

of $(\boldsymbol{s}_i, \boldsymbol{a}_i)$, R_{TD} is lower bounded by $(1-\gamma)\sum_i ||\nabla_\theta Q_\theta(\boldsymbol{x}_i)||_2^2$ and hence minimizing $R_{\text{TD}}(\theta)$ would minimize the feature norm instead of maximizing dot products. This also corresponds to the implicit regularizer we would obtain when training Q-functions via supervised learning and hence, our analysis predicts that offline SARSA with in-sample actions (i.e., when $(\boldsymbol{s}', \boldsymbol{a}') \in \mathcal{D}$) would behave similarly to supervised regression.

However, the regularizer behaves very differently when unseen state-action pairs $(\boldsymbol{s}'_i, \boldsymbol{a}'_i)$ appear only on the right-hand-side of the backup. This happens with any algorithm where $\boldsymbol{a}'$ is not the dataset action, which is the case for all deep RL algorithms that compute target values by selecting $\boldsymbol{a}'$ according to the current policy. In this case, we expect the dot product of gradients at $(\boldsymbol{s}, \boldsymbol{a})$ and $(\boldsymbol{s}', \boldsymbol{a}')$ to be large at any attractive fixed point, since this minimizes $R_{\text{TD}}(\theta)$. This is precisely a form of co-adaptation: *gradients at out-of-sample state-action tuples are highly similar to gradients at observed state-action pairs measured by the dot product*. This observation is also supported by the analysis in Section 6.2.1. Finally, note that the choice of M is a modelling assumption, and to derive our explicit regularizer, later in the paper, we will make a simplifying choice of M. However, we also empirically verify that a different choice of M, given by label noise, works well.

Why is implicit regularization detrimental to policy performance? To answer this question, we present theoretical and empirical evidence that illustrates the adverse effects of this implicit regularizer. Empirically, we ran two algorithms, DQN and CQL, initialized from a high-performing Q-function checkpoint, which attains relatively small feature dot products (i.e., the second term of $R_{\text{TD}}(\theta)$ is small). Our goal is to see if TD updates starting from such a "good" initialization still stay around it or diverge to poorer solutions. Our theoretical analysis in Section 6.2.2 would predict that TD learning would destabilize from such a solution, since it would not be a stable fixed point. Indeed, as shown in Figure 14, the policy immediately degrades, and the the dot-product similarities start to increase. This even happens with CQL, which explicitly corrects for distributional shift confounds, implying that the performance drop cannot be directly explained by the typical out-of-distribution action explanations. To investigate the reasons behind this drop, we also measured the training loss function values for these algorithms (i.e.,

TD error for DQN and TD error + CQL regularizer for CQL) and find in Figure 14 that the loss values are generally small for both CQL and DQN. This indicates that the preference to increase dot products is not explained by an inability to minimize TD error. In Appendix D.1.6, we show that this drop in performance when starting from good solutions can be effectively mitigated with our proposed Cal-QL explicit regularizer for both DQN and CQL. Thus we find that not only standard TD learning degrades from a good solution in favor of increasing feature dot products, but keeping small dot products enables these algorithms to remain stable near the good solution.

To motivate why co-adapted features can lead to poor performance in TD-learning, we study the convergence of linear TD-learning on co-adapted features. Our theoretical result characterizes a lower bound on the feature dot products in terms of the feature norms for state-action pairs in the dataset $\mathcal{D}$, which if satisfied, will inhibit convergence:

Proposition 6.2.2 (TD-learning on co-adapted features)**.** *Assume that the features* $\Phi = [\phi(\boldsymbol{s}, \boldsymbol{a})]_{\boldsymbol{s},\boldsymbol{a}}$ *are used for linear TD-learning. Then, if*

$$\sum_{\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}' \in \mathcal{D}} \phi(\boldsymbol{s}, \boldsymbol{a})^\top \phi(\boldsymbol{s}', \boldsymbol{a}') \geq \frac{1}{\gamma} \sum_{\boldsymbol{s},\boldsymbol{a} \in \mathcal{D}} \phi(\boldsymbol{s}, \boldsymbol{a})^\top \phi(\boldsymbol{s}, \boldsymbol{a}),$$

linear TD-learning using features Φ *will not converge.*

A proof of Proposition 6.2.2 is provided in Appendix D.4 and it relies on a stability analysis of linear TD. While features change during training for TD-learning with neural networks, and arguably linear TD is a simple model to study consequences of co-adapted features, even in this simple linear setting, Proposition 6.2.2 indicates that TD-learning may be non-convergent as a result of co-adaptation.

6.2.3 *DR3: Explicit Regularization for Deep TD-Learning*

Since the implicit regularization effects in TD-learning can lead to feature co-adaptation, which in turn is correlated with poor performance, can we instead derive an *explicit* regularizer to alleviate this issue? Inspired by the analysis in the previous section, we will propose an *explicit* regularizer that attempts to counteract the second term in Equation 6.2.3, which would otherwise lead to co-adaptation and poor representations. The *explicit* regularizer that offsets the difference between the two implicit regularizers is given by: $\Delta(\theta) = \sum_i \text{trace}\left[\Sigma_M^{*\top} \nabla_\theta Q_\theta(\boldsymbol{s}_i, \boldsymbol{a}_i) \nabla_\theta Q_\theta(\boldsymbol{s}'_i, \boldsymbol{a}'_i)^\top\right]$, which represents the second term of $R_{\text{TD}}(\theta)$. Note that we drop the stop gradient on $Q_\theta(\boldsymbol{s}'_i, \boldsymbol{a}'_i)$ in $\Delta(\theta)$, as it performs slightly better in practice (Table 39), although as shown in that Table, the version with the stop gradient also significantly improves over the base method. The first term of $R_{\text{TD}}(\theta)$ corresponds to the regularizer from supervised learning. Our proposed method, DR3, simply combines approximations to $\Delta(\theta)$ with various offline RL algorithms. For

any offline RL algorithm, ALG, with objective $\mathcal{L}_{\text{ALG}}(\theta)$, the training objective with DR3 is given by: $\mathcal{L}(\theta) := \mathcal{L}_{\text{ALG}}(\theta) + c_0\Delta(\theta)$, where c_0 is the DR3 coefficient. See Appendix D.5.3 for details on how we tune c_0 in this paper.

Practical version of DR3. In order to practically instantiate DR3, we need to choose a particular noise model M. In general, it is not possible to know beforehand the "correct" choice of M (Equation 6.2.2), even in supervised learning, as this is a complicated function of the data distribution, neural network architecture and initialization. Therefore, we instantiate DR3 with two heuristic choices of M: **(i)** M induced by label noise studied in prior work for supervised learning and for which we need to run another, separate fixed-point computation for M, and **(ii)** a simpler alternative that sets $\Sigma^*_M = I$. We find that both of these variants generally perform well empirically (Figure 18), and improve over the base offline RL method, and so we utilize **(ii)** in practice due to low computational costs. Additionally, because computing and backpropagating through per-example gradient dot products is slow, we instead approximate $\Delta(\theta)$ with the contribution only from the last layer parameters (*i.e.* $\sum_i \nabla_{\boldsymbol{w} Q_\theta(\boldsymbol{s}_i, \boldsymbol{a}_i)}{}^\top \nabla_{\boldsymbol{w} Q_\theta(\boldsymbol{s}'_i, \boldsymbol{a}'_i)}$), similarly to tractable Bayesian neural nets. As shown in Appendix D.1.4, the practical version of DR3 performs similarly to the label-noise version.

$$\textbf{Explicit DR3 regularizer}: \qquad \overline{\mathcal{R}}_{\text{exp}}(\theta) = \sum_{i \in \mathcal{D}} \phi(\boldsymbol{s}_i, \boldsymbol{a}_i)^\top \phi(\boldsymbol{s}'_i, \boldsymbol{a}'_i). \tag{6.2.6}$$

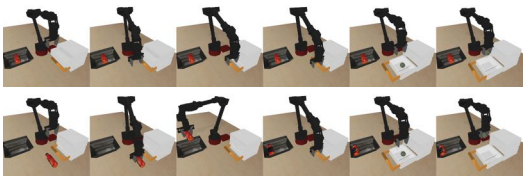
Feature normalization instead of regularization. In many problems, it may be more convenient to completely avoid the hyperparameter that weights the explicit DR3 regularizer from Equation 6.2.6 and yet attain a similar boost in performance. To this end, we propose to regularize the magnitude of the learned features of the state-action pair (or state) by introducing a "normalization" layer in the Q-network. This layer forces the learned features to have an ℓ_2 norm of 1 by construction. As we will discuss in the second part of this chapter, we found that this layer speeds up learning, resulting in better performance, without needing to tune the hyperparameter associated with the regularization variant of DR3.

6.2.4 *Experimental Evaluation of DR3*

Our experiments aim to evaluate the extent to which DR3 improves performance in offline RL in practice, and to study its effect on prior observations of rank collapse. To this end, we investigate if DR3 improves offline RL performance and stability on three offline RL benchmarks: Atari 2600 games with discrete actions [6], continuous control tasks from D4RL [92], and image-based robotic manipulation tasks [303]. Following prior work [92, 116], we evaluate DR3 in terms of final offline RL performance after a given number of iterations. Additionally, we report *training stability*, which is important

in practice as offline RL does not admit cheap validation of trained policies for model selection. To evaluate stability, we train for a large number of gradient steps (2-3x longer than prior work) and either report the **average performance** over the course of training or the final performance at the end of training. We expect that a stable method that does not unlearn with more gradient steps, should have better average performance, as compared to a method that attains good peak performance but degrades with more training. See Appendix D.5 for further details.

Offline RL on Atari 2600 games. We compare DR3 to prior offline RL methods on a set of offline Atari datasets of varying sizes and quality, akin to Agarwal et al. [6], Kumar et al. [182]. We evaluated on three datasets: **(1)** 1% and 5% samples drawn uniformly at random from DQN replay; **(2)** a dataset with more suboptimal data consisting of the first 10% samples observed by an online DQN. Following Agarwal et al. [7], we report the interquartile mean (IQM) normalized scores across 17 games over the course of training in Figure 16 and report the IQM average performance in Table 8. Observe that combining DR3 with modern offline RL methods (CQL, REM) attains the best final and average performance across the 17 Atari games tested on, directly improving upon prior methods across all the datasets. When DR3 is used in conjunction with REM, it prevents severe unlearning and performance degradation with more training. CQL + DR3 improves by **20%** over CQL on final performance and attains **25%** better average performance. While DR3 is not unequivocally "stable", as its performance also degrades relative to the peak it achieves (Figure 16), it is more stable relative to base offline RL algorithms. We also compare DR3 to the srank(Φ) penalty proposed to counter rank collapse [182]. Directly taking median normalized score improvements reported by Kumar et al. [182], CQL + DR3 improves by over **2x** (31.5%) over naïve CQL relative to the srank penalty (14.1%), indicating DR3's efficacy.



Offline RL on robotic manipulation from images. Next, we aim to evaluate the efficacy of DR3 on two image-based robotic manipulation tasks [303] (visualized on the right) that require composition of skills (e.g., opening a drawer, closing a drawer, picking an obstructive object, placing an object, etc.) over extended horizons using only a sparse 0-1 reward. As shown in Figure 15, combining DR3 with COG improves over COG.

DR3 does not suffer from rank collapse. Prior work [182] has shown that implicit regularization can lead to a rank collapse issue in TD-learning, preventing Q-networks from using full capacity. To see if DR3 addresses the rank collapse issue, we follow Kumar et al. [182] and plot the effective rank of learned features with DR3 in Figure 17 (DQN, REM in Appendix D.1.3). While the value of the effective rank decreases during training with naïve bootstrapping, we find that rank of DR3 features typically does not collapse,

Table 8: IQM normalized average performance (training stability) across 17 games, with 95% CIs in parenthesis, after 6.5M gradient steps for the 1% setting and 12.5M gradient steps for the 5%, 10% settings. Individual performances reported in Kumar et al. [183]. DR3 improves the stability over both CQL and REM.

Data	CQL	CQL + DR3	REM	REM + DR3
1%	43.7 (39.6, 48.6)	**56.9** (52.5, 61.2)	4.0 (3.3, 4.8)	**16.5** (14.5, 18.6)
5%	78.1 (74.5, 82.4)	**105.7** (101.9, 110.9)	25.9 (23.4, 28.8)	**60.2** (55.8, 65.1)
10%	59.3 (56.4, 61.9)	**65.8** (63.3, 68.3)	53.3 (51.4, 55.3)	**73.8** (69.3, 78)

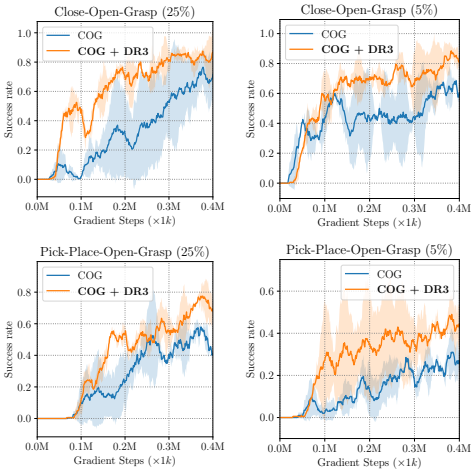


Figure 15: Performance of DR3 + COG on two manipulation tasks using only 5% and 25% of the data used by Singh et al. [303] to make these more challenging. COG + DR3 outperforms COG in training and attains higher average and final performance.

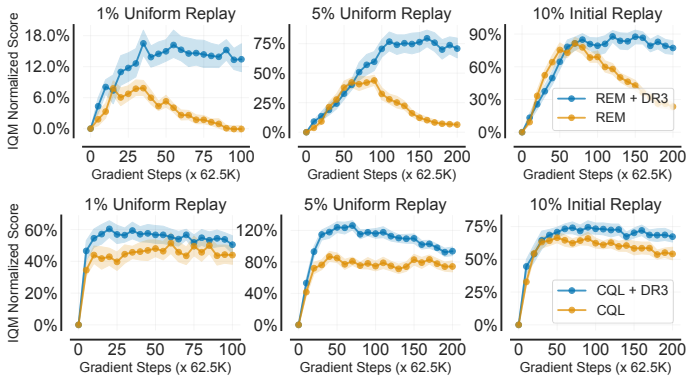


Figure 16: Normalized performance across 17 Atari games for REM + DR3 (top), CQL + DR3 (bottom). x-axis represents *gradient steps*; no new data is collected. While naïve REM suffers from a degradation in performance with more training, REM + DR3 not only remains generally stable with more training, but also attains higher final performance. CQL + DR3 attains higher performance than CQL. We report IQM with 95% stratified bootstrap CIs [7].

despite no explicit term encouraging this. Finally, we test the robustness/sensitivity of each layer in the learned Q-network to re-initialization [372] during training and find that DR3 alters the features to behave similarly to supervised learning (Figure 22).

Comparing explicit regularizers for different choices of noise covariance M**.** Finally, we investigate the behavior of different implicit regularizers derived via two choices of M in Equation 6.2.3 and the corresponding explicit regularizers. While the explicit regularizer we use in practice is a simplifying choice that works well, another choice of M is the covariance matrix induced by label noise, which requires explicit computation of Σ^*_M. Observe in Figure 18 that the explicit regularizer for our simplifying choice is not worse than the different choice of M. This justifies utilizing our simplified, heuristic choice of setting $\Sigma^*_M = I$ in practice. Results on five Atari games are shown in Appendix D.1.4.

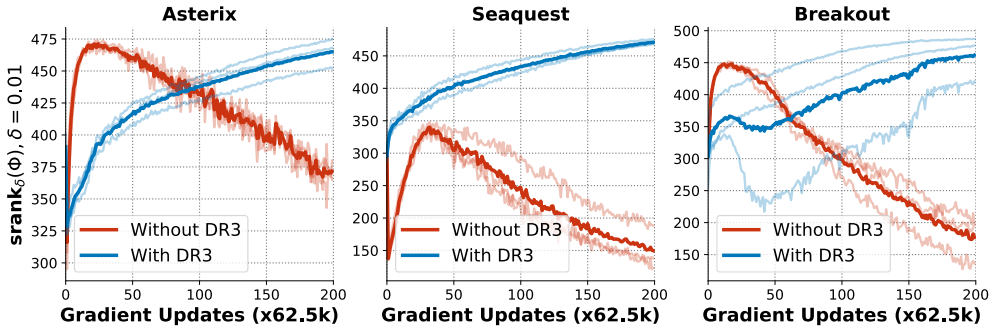


Figure 17: Trend of effective rank, srank(Φ) of features Φ learned by the Q-function when trained with TD error (red, "Without DR3") and with TD error + DR3 (blue, "With DR3") on three Atari games using the 5% dataset. Note that DR3 alleviates rank collapse, without explicitly aiming to.

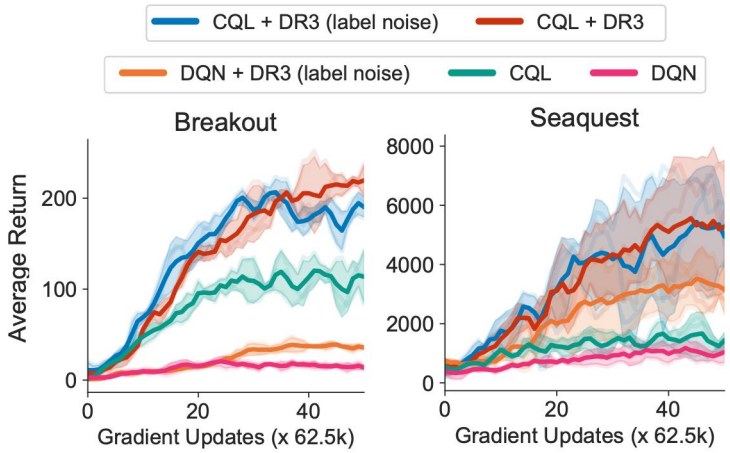


Figure 18: Comparing DR3 for our simplifying choice of M, and M induced by label noise, with base CQL and DQN algorithms. Note that both of these penalties when applied over CQL improve performance.

6.3 Scaled Q-learning: large-scale study of offline Q-learning

Our goal in this section is to learn a single policy that is effective at multiple Atari games and can be fine-tuned to new games. For training, we utilize the set of 40 Atari games used by Lee et al. [199], and for each game, we utilize the experience collected in the DQN-Replay dataset [6] as our offline dataset. We consider two different dataset compositions:

1. **Sub-optimal** dataset consisting of the initial 20% of the trajectories (10M transitions) from DQN-Replay for each game, containing 400 million transitions overall with average human-normalized interquartile-mean (IQM) [8] score of 51%. Since this dataset does not contain optimal trajectories, we do not expect methods that simply copy behaviors in this dataset to perform well. Instead, we would expect methods that can combine useful segments of sub-optimal trajectories to perform well.

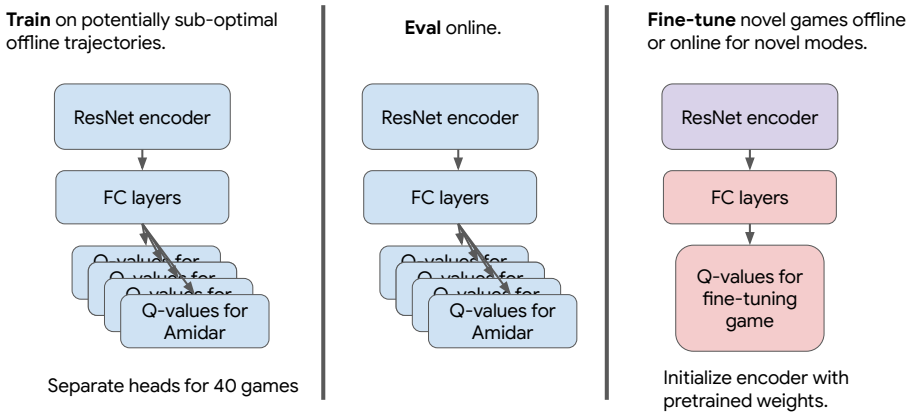


Figure 19: An overview of the training and evaluation setup. Models are trained offline with potentially sub-optimal data. We adapt CQL to the multi-task setup via a multi-headed architecture. The pre-trained visual encoder is reused in fine-tuning (the weights are either frozen or fine-tuned), whereas the downstream fully-connected layers are reinitialized and trained.

2. **Near-optimal** dataset, used by Lee et al. [199], consisting of all the experience (50M transitions) encountered during training of a DQN agent including human-level trajectories, containing 2 billion transitions with average human-normalized IQM score of 93.5%.

Evaluation. We evaluate our method in a variety of settings as we discuss in our experiments in Section 6.3.2. Due to excessive computational requirements of running huge models, we are only able to run our main experiments with one seed. Prior work [199] that also studied offline multi-game Atari evaluated models with only one seed. That said, to ensure that our evaluations are reliable, for reporting performance, we follow the recommendations by Agarwal et al. [8]. Specifically, we report interquartile mean (IQM) normalized scores, which is the average scores across middle 50% of the games, as well as performance profiles for qualitative summarization.

6.3.1 *Our Approach for Scaling Offline RL*

In this section, we describe the important architectural design decisions required to make offline CQL effective in learning highly-expressive neural network policies from large offline datasets.

Parameterization of Q-values and TD error. In the single game setting, both mean-squared TD error and distributional TD error perform comparably online [8] and offline [181, 183]. In contrast, we observed, perhaps surprisingly, that mean-squared TD error does not scale well, and performs much worse than using a **categorical distributional representation of return values** [24] when we train on many Atari games. We hypothesize that this is because even with reward clipping, Q-values for different games often span different ranges, and training a single network with shared parameters to

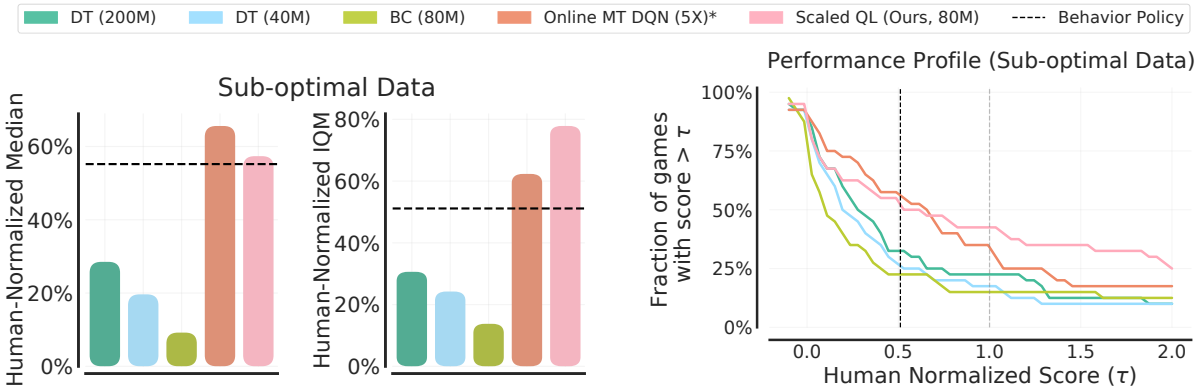


Figure 20: Offline multi-task performance on 40 games with sub-optimal data. **Left**. Scaled QL significantly outperforms the previous state-of-the-art method, DT, attaining about a **2.5x** performance improvement in normalized IQM score. To contextualize the absolute numbers, we include online multi-task Impala DQN [74] trained on 5x as much data. **Right**. Performance profiles [8] showing the distribution of normalized scores across all 40 training games (higher is better). Scaled QL stochastically dominates other offline RL algorithms and achieves superhuman performance in 40% of the games. "Behavior policy" corresponds to the score of the dataset trajectories. Online MT DQN (5X), taken directly from Lee et al. [199], corresponds to running multi-task online RL for 5x more data with IMPALA (details in Appendix D.8.5).

accurately predict all of them presents challenges pertaining to gradient interference along different games [137, 363]. While prior works have proposed to use adaptive normalization schemes [137, 189], preliminary experiments with these approaches were not effective to close the gap.

Q-function architecture. Since large neural networks has been crucial for scaling to large, diverse datasets in NLP and vision [e.g., 317, 34, 165], we explore using bigger architectures for scaling offline Q-learning. We use standard feature extractor backbones from vision, namely, the Impala-CNN architectures [74] that are fairly standard in deep RL and ResNet 34, 50 and 101 models from the ResNet family [131]. We make modifications to these networks following certain recommendations from robotic RL (see Chapter 9 for a detailed discussion of these changes): we utilize group normalization instead of batch normalization in ResNets, and utilize point-wise multiplication with a learned spatial embedding when converting the output feature map of the vision backbone into a flattened vector which is to be fed into the feed-forward part of the Q-function.

To handle the multi-task setting, we use a multi-headed architecture where the Q-network outputs values for each game separately. The architecture uses a shared encoder and feedforward layers with separate linear projection layers for each game (Figure 21). The training objective of CQL is computed using the Q-values for the game that the transition originates from.

Feature Normalization via DR3. Our preliminary experiments with the design decisions discussed above on a subset of games did not attain good performance. In the single-

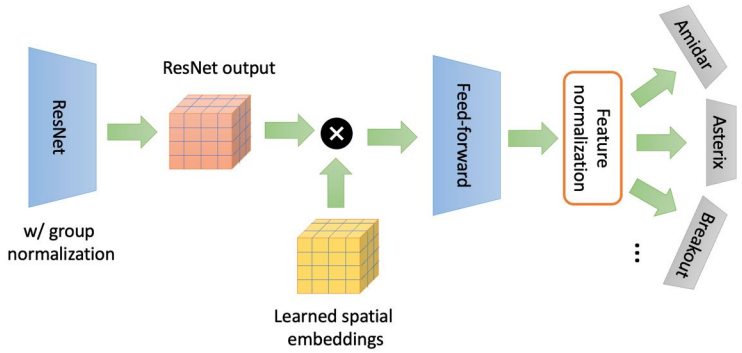


Figure 21: An overview of the network architecture. The key design decisions are: (1) the use of ResNet models with learned spatial embeddings and group normalization, (2) use of a distributional representation of return values and cross-entropy TD loss for training (i.e., C51 [24]), and (3) feature normalization to stablize training.

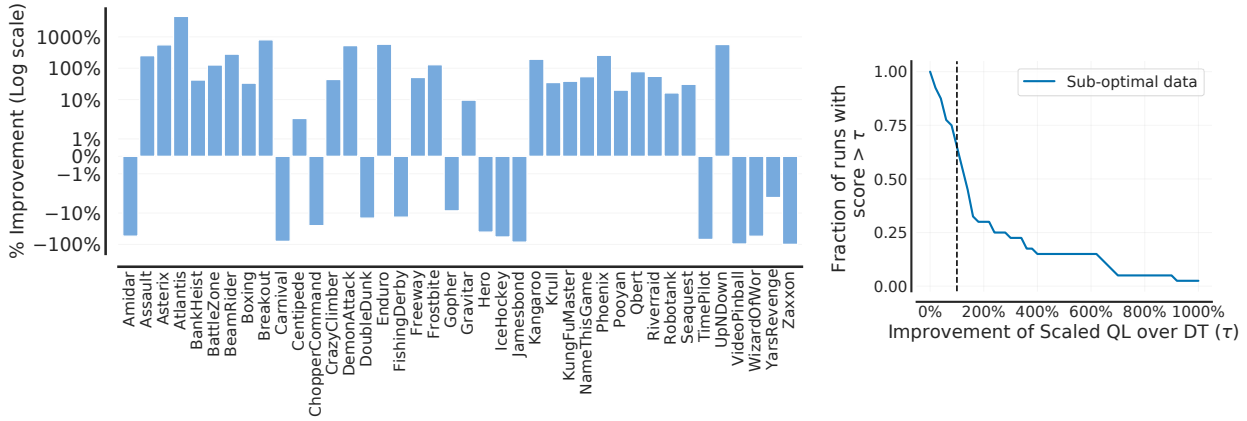


Figure 22: Comparing Scaled QL to DT on all training games on the sub-optimal dataset.

task setting, the DR3 regularizer that stabilizes training and allows the network to better use capacity, however, as we discussed in Section 6.2, it introduces an additional hyperparameter to tune. To alleviate this bottleneck, we utilize the normalization variant of DR3 for our large-scale experiments, and regularize the magnitude of the learned features of the observation to have an ℓ_2 norm of 1 by construction. We present an ablation study analyzing this choice in Table 10 and find that indeed DR3 normalization improves performance across the board.

To summarize, the primary modifications that enable us to scale CQL are: **(1)** use of large ResNets with learned spatial embeddings and group normalization, **(2)** use of a distributional representation of return values and cross-entropy loss for training (i.e., C51 [24]), and **(3)** feature normalization at intermediate layers via DR3 to prevent feature co-adaptation (Section 6.2). We call our approach **Scaled Q-learning**.

6.3.2 *Experimental Evaluation*

In our experiments, we study how our approach, scaled Q-learning, can simultaneously learn from sub-optimal and optimal data collected from 40 different Atari games. We compare the resulting multi-task policies to behavior cloning (BC) with same architecture as scaled QL, and the prior state-of-the-art method based on decision transformers (DT) [44], which utilize return-conditioned supervised learning with large transformers [199], and have been previously proposed for addressing this task. We also study the efficacy of the multi-task initialization produced by scaled Q-learning in facilitating rapid transfer to new games via both offline and online fine-tuning, in comparison to state-of-the-art self-supervised representation learning methods and other prior approaches. Our goal is to answer the following questions: **(1)** How do our proposed design decisions impact performance scaling with high-capacity models?, **(2)** Can scaled QL more effectively leverage higher model capacity compared to naïve instantiations of Q-learning?, **(3)** Do the representations learned by scaled QL transfer to new games? We will answer these questions in detail through multiple experiments in the coming sections, but we will first summarize our main results below.

Main empirical findings. Our main results are summarized in Figures 20 and 23. These figures show the performance of scaled QL, multi-game decision transformers [199] (marked as "DT"), a prior method based on supervised learning via return conditioning, and standard behavioral cloning baselines (marked as "BC") in the two settings discussed previously, where we must learn from: (i) near optimal data, and (ii) sub-optimal data obtained from the initial 20% segment of the replay buffer (see our discussion about the problem setup). See Figure 22 for a direct comparison between DT and BC.

In the more challenging sub-optimal data setting, scaled QL attains a performance of **77.8%** IQM human-normalized score, although trajectories in the sub-optimal training dataset only attain 51% IQM human-normalized score. Scaled QL also outperforms the prior DT approach by **2.5 times** on this dataset, even though the DT model has more than twice as many parameters and uses data augmentation, compared to scaled QL.

In the 2^{nd} setting with near-optimal data, where the training dataset already contains expert trajectories, scaled QL with 80M parameters still outperforms the DT approach

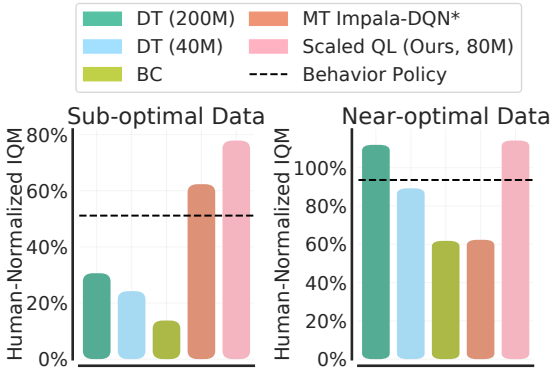


Figure 23: Offline scaled conservative Q-learning vs other prior methods with near-optimal data and sub-optimal data. Scaled QL outperforms the best DT model, attaining an IQM human-normalized score of **114.1%** on the near-optimal data and **77.8%** on the sub-optimal data, compared to 111.8% and 30.6% for DT, respectively.

with 200M parameters, although the gap in performance is small (3% in IQM performance, and 20% on median performance). Overall, these results show that scaled QL is an effective approach for learning from large multi-task datasets, for a variety of data compositions including sub-optimal datasets, where we must stitch useful segments of suboptimal trajectories to perform well, and near-optimal datasets, where we should attempt to mimic the best behavior in the offline dataset.

To the best of our knowledge, these results represent the largest performance improvement over the average performance in the offline dataset on such a challenging problem. We will now present experiments that show that offline Q-learning scales and generalizes.

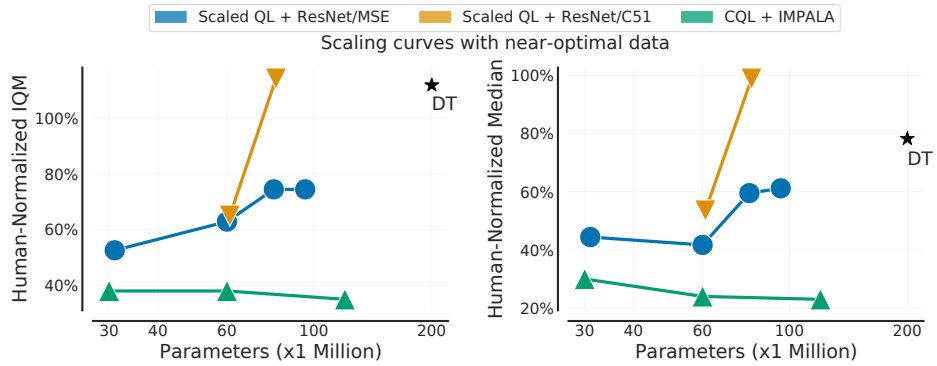


Figure 24: Scaling trends for offline Q-learning. Observe that while the performance of scaled QL instantiated with IMPALA architectures [74] degrades as we increase model size, the performance of scaled QL utilizing the ResNets described in Section 6.3.1 continues to increase as model capacity increases. This is true for both an MSE-style TD error as well as for the categorical TD error used by C51 (which performs better on an absolute scale). The CQL + IMPALA performance numbers are from [199].

6.3.2.1 *Does Offline Conservative Q-Learning Scale Favorably?*

One of the primary goals of this chapter was to understand if scaled Q-learning is able to leverage the benefit of higher capacity architectures. Recently, Lee et al. [199] found that the performance of CQL with the IMPALA architecture does not improve with larger model sizes and may even degrade with larger model sizes. To verify if scaled Q-learning can address this limitation, we compare our value-based offline RL approach with a variety of model families: **(a)** IMPALA family [74]: three IMPALA models with varying widths $(4, 8, 16)$ whose performance numbers are taken directly from Lee et al. [199] (and was consistent with our preliminary experiments), **(b)** ResNet 34, 50, 101 and 152 from the ResNet family, modified to include group normalization and learned spatial embeddings.These architectures include both small and large networks, spanning a wide range from 1M to 100M parameters. As a point of reference, we use the scaling trends of the multi-game decision transformer and BC approaches from Lee et al. [199].

Observe in Figure 24 that the performance of scaled Q-learning improves as the underlying Q-function model size grows. Even though the standard mean-squared error formulation

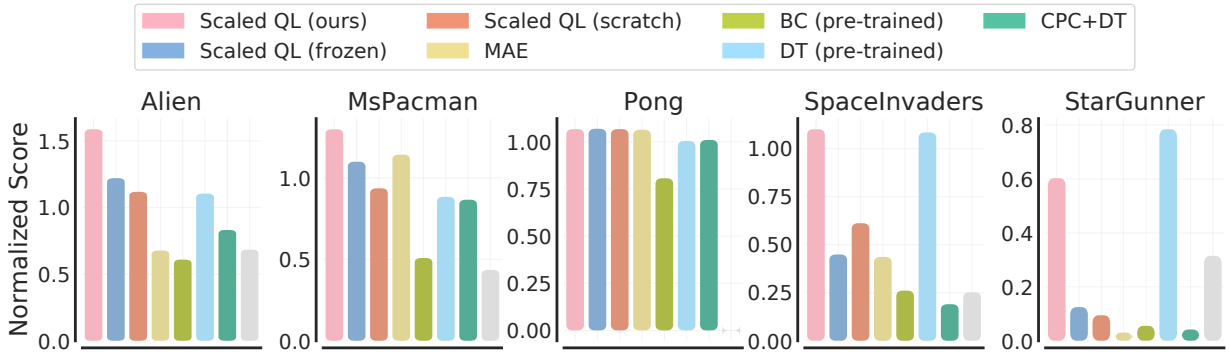


Figure 25: Offline fine-tuning performance on unseen games trained with 1% of held-out game's data, measured in terms of DQN-normalized score, following [199]. On average, pre-training with scaled QL outperforms other methods by **82%**. Furthermore, scaled QL improves over scaled QL (scratch) by 45%, indicating that the representations learned by scaled QL during multi-game pre-training are useful for transfer. Self-supervised representation learning (CPC, MAE) alone does not attain good fine-tuning performance.

of TD error results in worse absolute performance than C51 (blue vs orange), for both of these versions, the performance of scaled Q-learning increases as the models become larger. This result indicates that value-based offline RL methods can scale favorably, and give rise to better results, but this requires carefully picking a model family. This also explains the findings from Lee et al. [199]: while this prior work observed that CQL with IMPALA scaled poorly as model size increases, they also observed that the performance of return-conditioned RL instantiated with IMPALA architectures also degraded with higher model sizes. Combined with the results in Figure 24 above, this suggests that poor scaling properties of offline RL can largely be attributed to the choice of IMPALA architectures, which may not work well in general even for supervised learning methods (like return-conditioned BC).

6.3.2.2 *Can Offline RL Learn Useful Initializations that Enable Fine-Tuning?*

Next, we study how multi-task training on multiple games via scaled QL can learn general-purpose representations that can enable *rapid* fine-tuning to new games. We study this question in two scenarios: fine-tuning to a new game via offline RL with a small amount of held-out data (1% uniformly subsampled datasets from DQN-Replay [6]), and finetuning to a new game mode via sample-efficient online RL initialized from our multi-game offline Q-function. For finetuning, we transfer the weights from the visual encoder and reinitialize the downstream feed-forward component (Figure 19). For both of these scenarios, we utilize a ResNet101 Q-function trained via the methodology in Section 6.3.1, using C51 and feature normalization.

Scenario 1 (Offline fine-tuning): First, we present the results for fine-tuning in an offline setting: following the protocol from Lee et al. [199], we use the pre-trained representations to rapidly learn a policy for a novel game using limited offline data

(1% of the experience of an online DQN run). In Figure 25, we present our results for offline fine-tuning on 5 games from Lee et al. [199], Alien, MsPacman, Space Invaders, StarGunner and Pong, alongside the prior approach based on decision transformers ("DT (pre-trained)"), and fine-tuning using pre-trained representations learned from state-of-the-art self-supervised representation learning methods such as contrastive predictive coding (CPC) [252] and masked autoencoders (MAE) [130]. For CPC performance, we use the baseline reported in Lee et al. [199]. MAE is a more recent self-supervised approach that we find generally outperformed CPC in this comparison. For MAE, we first pretrained a vision transformer (ViT-Base) [65] encoder with 80M parameters trained via a reconstruction loss on observations from multi-game Atari dataset and freeze the encoder weights as done in prior work [346]. Then, with this frozen visual encoder, we used the same feed forward architecture, Q-function parameterization, and training objective (CQL with C51) as scaled QL to finetune the MAE network. We also compare to baseline methods that do not utilize any multi-game pre-training (DT (scratch) and Scaled QL (scratch)).

Results. Observe in Figure 25 that multi-game pre-training via scaled QL leads to the best fine-tuning performance and improves over prior methods, including decision transformers trained from scratch. Importantly, we observe *positive transfer* to new games via scaled QL. Prior works [18] running multi-game Atari (primarily in the online setting) have generally observed negative transfer across Atari games. We show for the first time that pre-trained representations from Q-learning enable positive transfer to novel games that significantly outperforms return-conditioned supervised learning methods and dedicated representation learning approaches.

Scenario 2 (Online fine-tuning): Next, we study the efficacy of the learned representations in enabling online fine-tuning. While deep RL agents on ALE are typically trained on default game modes (referred to as $m0d0$), we utilize new *variants* of the ALE games designed to be challenging for humans [223] for online-finetuning. We investigate whether multi-task training on the 40 default game variants can enable fast online adaptation to these never-before-seen variants. In contrast to offline fine-tuning (Scenario 1), this setting tests whether scaled QL can also provide a good initialization for online data collection and learning, for closely related but different tasks. Following Farebrother et al. [80], we use the same *variants* investigated in this prior work: Breakout, Hero, and Freeway, which we visualize in Figure 26 (left). To disentangle the performance gains from multi-game pre-training and the choice of Q-function architecture, we compare to a baseline approach ("scaled QL (scratch)") that utilizes an identical Q-function architecture as pre-trained scaled QL, but starts from a random initialization. As before, we also evaluate fine-tuning performance using the representations obtained via masked auto-encoder pre-training [130, 346]. We also compare to a single-game DQN performance attained

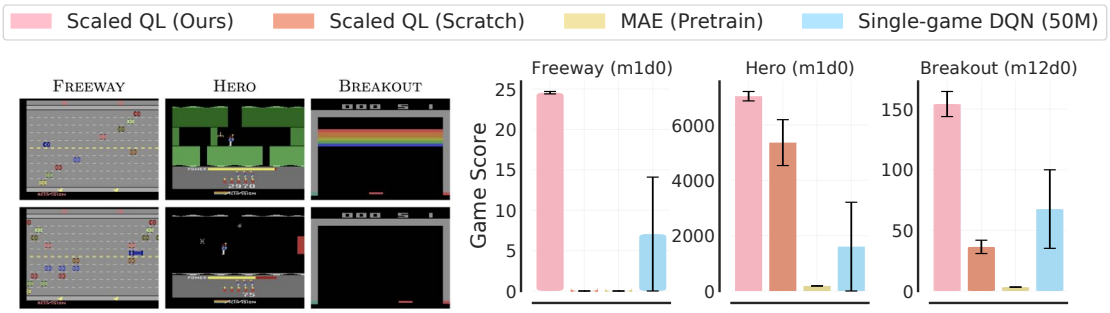


Figure 26: Online fine-tuning results on unseen game *variants*. **Left**. The top row shows default variants and the bottom row shows unseen variants evaluated for transfer: Freeway's mode 1 adds buses, more vehicles, and increases velocity; Hero's mode 1 starts the agent at level 5; Breakout's mode 12 hides all bricks unless the ball has recently collided with a brick. **Right**. We fine-tune all methods except single-game DQN for 3M online frames (as we wish to test fast online adaptation). Error bars show minimum and maximum scores across 2 runs while the bar shows their average. Observe that scaled QL significantly outperforms learning from scratch and single-game DQN with 50M online frames. Furthermore, scaled QL also outperforms RL fine-tuning on representations learned using masked auto-encoders. See Figure 34 for learning curves.

after training for 50M steps, 16× more transitions than what is allowed for scaled QL, as reported by Farebrother et al. [80].

Results. Observe in Figure 26 that fine-tuning from the multi-task initialization learned by scaled QL significantly outperforms training from scratch as well as the single-game DQN run trained with **16x** more data. Fine-tuning with the frozen representations learned by MAE performs poorly, which we hypothesize is due to differences in game dynamics and subtle changes in observations, which must be accurately accounted for in order to learn optimal behavior [57]. Our results confirm that offline Q-learning can both effectively benefit from higher-capacity models and learn multi-task initializations that enable sample-efficient transfer to new games.

6.3.2.3 *Ablation Studies*

Finally, in this section we perform controlled ablation studies to understand how crucial the design decisions introduced in Section 6.3.1 are for the success of scaled Q-learning. In particular, we will attempt to understand the benefits of using C51 and feature normalization.

MSE vs C51: We ran scaled Q-learning with identical network architectures (ResNet 50 and ResNet 101), with both the conventional squared error formulation of TD error, and compare it to C51, which our main results utilize. Observe in Table 9 that C51 leads to much better performance for both ResNet 50 and ResNet 101 models. The boost in performance is the largest for ResNet 101, where C51 improves by over **39%** as measured by median human-normalized score. This observation is surprising since prior work [8] has shown that C51 performs on par with standard DQN with an Adam optimizer, which

Table 9: Performance of Scaled QL with the standard mean-squared TD-error and C51 in the offline 40-game setting aggregated by the median human-normalized score. Observe that for both ResNet 50 and ResNet 101, utilizing C51 leads to a drastic improvement in performance.

	Scaled QL (ResNet 50)	Scaled QL (ResNet 101)
with MSE	41.1%	59.5%
with C51	53.5% (+12.4%)	98.9% (+39.4%)

all of our results use. One hypothesis is that this could be the case as TD gradients would depend on the scale of the reward function, and hence some games would likely exhibit a stronger contribution in the gradient. This is despite the fact that our implementation of MSE TD-error already attempts to correct for this issue by applying the unitary scaling technique from [189] to standardize reward scales across games. That said, we still observe that C51 performs significantly better.

Importance of feature normalization: We ran small-scale experiments with and without feature normalization (Section 6.3.1). In these experiments, we consider a multi-game setting with only 6 games: Asterix, Breakout, Pong, SpaceInvaders, Seaquest, and we train with the initial 20% data for each game. We report aggregated median human-normalized score across the 6 games in Table 10 for three different network architectures (ResNet 34, ResNet 50 and ResNet 101). Observe that the addition of feature normalization significantly improves performance for all the models. Motivated by this initial empirical finding, we used feature normalization in all of our main experiments. Overall, the above ablation studies validate the efficacy of the two key design decisions in this chapter.

Table 10: Performance of Scaled QL with and without feature normalization in the 6 game setting reported in terms of the median human-normalized score. Observe that with models of all sizes, the addition of feature normalization improves performance.

	Scaled QL (ResNet 34)	Scaled QL (ResNet 50)	Scaled QL (ResNet 101)
without feature normalization	50.9%	73.9%	80.4%
with feature normalization	78.0% (+28.9%)	83.5% (+9.6%)	98.0% (+17.6%)

Additional ablations: We also conducted ablation studies for the choice of the backbone architecture (spatial learned embeddings) in Appendix D.7.2, and observed that utilizing spatial embeddings is better. We also evaluated the performance of scaled QL without conservatism to test the importance of utilizing pessimism in our setting with diverse data in Appendix D.7.3, and observe that pessimism is crucial for attaining good performance on an average. We also provide some scaling studies for another offline RL method (discrete BCQ) in Appendix D.7.1.

6.3.3 *Related Work*

Prior works have sought to train a single generalist policy to play multiple Atari games simultaneously from environment interactions, either using off-policy RL with online data collection [74, 136, 307], or policy distillation [320, 280] from single-task policies. While our work also focuses on learning such a generalist multi-task policy, it investigates whether we can do so by scaling offline Q-learning on suboptimal offline data, analogous to how supervised learning can be scaled to large, diverse datasets. Furthermore, prior attempts to apply transfer learning using RL-learned policies in ALE [280, 256, 231] are restricted to a dozen games that tend to be similar and generally require an "expert", instead of learning how to play all games concurrently.

Closely related to our work, recent work train Transformers [330] on purely offline data for learning such a generalist policy using supervised learning (SL) approaches, namely, behavioral cloning [276] or return-conditioned behavioral cloning [199]. While these works focus on large datasets containing expert or near-human performance trajectories, our work focuses on the regime when we only have access to highly diverse but sub-optimal datasets. We find that these SL approaches perform poorly with such datasets, while offline Q-learning is able to substantially extrapolate beyond dataset performance (Figure 20). Even with near-optimal data, we observe that scaling up offline Q-learning outperforms SL approaches with 200 million parameters using as few as half the number of network parameters (Figure 24).

6.4 Discussion and Limitations

In the first part of this chapter, we attempted to characterize the implicit preference of TD-learning towards solutions that maximally co-adapt gradients (or features) at consecutive state-action tuples that appear in Bellman backup. This regularization effect is exacerbated when out-of-sample state-action samples are used for the Bellman backup and it can lead to poor policy performance. Inspired by the theory, we propose a practical explicit regularizer, DR3, that yields substantial improvements in stability and performance on a wide range of offline RL problems. We believe that this sort of an understanding the learning dynamics of offline deep Q-learning will lead to more robust and stable deep RL algorithms and enable predicting such instability issues, well in advance, which can inspire cross-validation and model selection strategies. This is an important, open challenge in offline RL, for which existing off-policy evaluation techniques are not practically sufficient [93, 184]. A limitation of our analysis of implicit regularization stems from the simplifying assumptions needed to tractably analyze the learning dynamics. For instance, our analysis holds in the neighborhood of fixed points of the temporal-difference backup, but it does not comment the learning dynamics when

the Q-function is far away from a fixed point. We hypothesize that this limitation can be addressed by building on techniques in Damian et al. [52] (or its follow-ups).

The second part of this chapter shows, for the first time (to the best of our knowledge), that offline conservative Q-learning can scale to high-capacity models when trained on large, diverse datasets. By scaling up capacity, we unlocked analogous trends to those observed in vision and NLP. We found that scaled Q-learning trains policies that exceed the average dataset performance and prior methods, especially when the dataset does not contain expert trajectories. Furthermore, by training a large-capacity model on diverse tasks, we show that Q-learning is sufficient to recover general-purpose representations that enable rapid learning of novel tasks. Although we described an approach that is sufficient to scale Q-learning, this is by no means optimal. The scale of the experiments limited the number of alternatives we could explore, and we expect that future work will greatly improve performance. For example, contrary to decision transformers (DT) [199], we did not use data augmentation in our experiments, which we believe can provide significant benefits. While we did a preliminary attempt to perform online fine-tuning on an entirely new game (SpaceInvaders), we found that this did not work well for any of the pretrained representations (see Figure 34). Addressing this is an important direction for future work. We speculate that this challenge is related to designing methods for learning better exploration from offline data, which is not required for offline fine-tuning. Overall, we believe that scaled Q-learning could serve as a starting point for RL-trained foundation models.

acknowledgements and funding

We thank Dibya Ghosh, Xinyang Geng, Dale Schuurmans, Marc Bellemare, Pablo Castro, Ofir Nachum, Ross Goroshin and Aleksandra Faust for informative discussions. We thank Sherry Yang, Ofir Nachum, and Kuang-Huei Lee for help with the multi-game decision transformer codebase; Anurag Arnab for help with the Scenic ViT codebase. We thank Zoubin Ghahramani and Douglas Eck for providing compute support. We thank the members of RAIL at UC Berkeley for their support and suggestions. We thank anonymous reviewers for feedback on an early version of this paper. This research is funded in part by the DARPA Assured Autonomy Program and in part, by compute resources from Microsoft Azure and Google Cloud. TM acknowledges support of Google Faculty Award, NSF IIS 2045685, the Sloan fellowship, and JD.com.

7

OFFLINE RL WITH MULTI-TASK AND UNLABELED DATA

Abstract

A natural use case of offline RL is in settings where we can pool large amounts of data collected in various scenarios for solving different tasks, and utilize all of this data to learn behaviors for all the tasks more effectively rather than training each one in isolation. As we discussed in Chapter 6, part of the approach for this is via *parameter sharing*, i.e., training a single multi-task policy and value function for solving all scenarios or tasks together. A complementary question in multi-task offline RL is that of *data sharing*: can we reroute data from one task to improve performance on the other tasks, when various tasks in a multi-task problem share the underlying dynamics? It has been noted that data sharing performs surprisingly poorly in practice. In this chapter, we analyze data sharing empirically and find that sharing data can actually exacerbate the distributional shift between the learned policy and the dataset, which in turn can lead to divergence of the learned policy and poor performance. To address this challenge, we develop a simple technique for data-sharing in multi-task offline RL that routes data based on the improvement over the task-specific data. We call this approach conservative data sharing (CDS), and show it can be applied with multiple single-task offline RL methods. We further extend CDS to the setting when the reward labels for different tasks cannot be computed during rerouting and show that it is nonetheless effective in utilizing data across tasks.

7.1 INTRODUCTION

Many realistic settings where we might want to apply offline RL involve inherently *multi-task* problems, where we seek to solve multiple tasks using all available data. For example, if our goal is to enable robots to acquire a range of different behaviors, it is more practical to collect a modest amount of data for each desired behavior, resulting in a large but heterogeneous dataset, rather than requiring a large dataset for every individual skill. Indeed, many existing datasets in robotics [83, 53, 293] and offline

RL [92] follow precisely this approach. Unfortunately, leveraging such heterogeneous datasets presents us with two unenviable choices. On one hand, we could train each task only on data collected specifically for that task, but such small datasets may be inadequate for achieving good performance. On the other hand, we could combine all the data together and use data relabeled from other tasks to improve offline training, but this poses two crucial challenges.

First, incorporating relabeled data from other tasks into standard offline RL algorithms would require labeling large datasets with rewards, which can be costly, especially if human labelers must provide these rewards. Second, even if the functional form of the reward function for the task of interest is known beforehand, and data from other tasks are labeled with accurate reward annotations, na"ively using all this data for training can actually often degrade performance compared to simple single-task training in practice [162]. To address these issues, in this chapter, we aim to study data sharing with and without reward annotations. Specifically, we aim to understand how data sharing affects RL performance in the offline RL setting and develop a reliable and effective method for selectively sharing data across tasks (Section 7.2). Additionally, we aim to tackle the problem of using unlabeled data and devise an approach that still allows us to benefit from diverse, unlabeled datasets, even when reward annotations cannot be inferred correctly (Section 7.3).

The primary contributions of Section 7.2 are an analysis of data sharing in offline multi-task RL and a new algorithm, *conservative data sharing* (CDS), for multi-task offline RL problems. CDS relabels a transition into a given task only when it is expected to improve performance based on a conservative estimate of the Q-function. After data sharing, similarly to prior offline RL methods, CDS applies a standard conservative offline RL algorithm, such as CQL [181] or BRAC [343], a policy-constraint offline RL algorithm. Further, we theoretically analyze CDS and characterize scenarios under which it provides safe policy improvement guarantees. Finally, we conduct extensive empirical analysis of CDS on multi-task locomotion, multi-task robotic manipulation with sparse rewards, multi-task navigation, and multi-task imaged-based robotic manipulation. We compare CDS to vanilla offline multi-task RL without sharing data, to naïvely sharing data for all tasks, and to existing data relabeling schemes for multi-task RL. CDS is the only method to attain good performance across all of these benchmarks, often significantly outperforming the best *domain-specific* method, improving over the next best method on each domain by **17.5%** on average.

The primary contribution of Section 7.3 is the demonstration that simply labeling unlabeled data with a reward of zero for use in multi-task offline RL is surprisingly effective in many cases. We perform extensive theoretical and empirical analysis to study conditions under which this simple approach, *unlabeled data sharing* (UDS), would either excel or fail,

and we study how reweighting the relabeled data with CDS (while still using zero reward as the label) can substantially increase the range of settings when UDS is successful. Our empirical evaluation, conducted over various multi-task offline RL scenarios such as locomotion, robotic manipulation from visual inputs, and ant-maze navigation shows that this simple zero reward strategy leads to improved performance, especially when combined with the CDS approach, even compared to methods that use more sophisticated learning and label propagation strategies to infer rewards. Overall, we demonstrate that an effective way to perform data sharing with unlabeled multi-task datasets in offline RL is to use a combination of CDS and UDS approaches.

7.2 Data Sharing for Multi-Task Offline Reinforcement Learning

7.2.1 *Related Work*

Multi-task RL algorithms. Multi-task RL algorithms [339, 256, 320, 74, 137, 363, 353, 357, 162, 306] focus on solving multiple tasks jointly in an efficient way. While multi-task RL methods seem to provide a way to build general-purpose agents [162], prior works have observed major challenges in multi-task RL, in particular, the optimization challenge [137, 287, 363]. Beyond the optimization challenge, how to perform effective representation learning via weight sharing is another major challenge in multi-task RL. Prior works have considered distilling per-task policies into a single policy that solves all tasks [280, 320, 107, 353], separate shared and task-specific modules [61], and incorporating additional supervision [306]. Finally, sharing data across tasks emerges as a challenge in multi-task RL, especially in the off-policy setting, as naïvely sharing data across all tasks turns out to hurt performance in certain scenarios [162]. Unlike most of these prior works, we focus on the offline setting where the challenges in data sharing are most relevant. Methods that study optimization and representation learning issues are complementary and can be readily combined with our approach.

Data sharing in multi-task RL. Prior works [10, 155, 265, 285, 76, 205, 162, 41] have found it effective to reuse data across tasks by recomputing the rewards of data collected for one task and using such relabeled data for other tasks, which effectively augments the amount of data available for learning each task and boosts performance. These methods perform relabeling either uniformly [162] or based on metrics such as estimated Q-values [76, 205], domain knowledge [162], the distance to states or images in goal-conditioned settings [10, 265, 242, 216, 309, 214, 142, 220, 356, 41], and metric learning for robust inference in the offline meta-RL setting [207]. All of these methods either require online data collection and do not consider data sharing in a fully offline setting, or only consider offline goal-conditioned or meta-RL problems [41, 207]. While these prior works empirically find that data sharing helps, we believe that our analysis in Section 7.2.3 provides the first analytical understanding of why and when data sharing can help in multi-task offline RL and why it hurts in some cases. Specifically, our analysis reveals

the effect of distributional shift introduced during data sharing, which is not taken into account by these prior works. Our proposed approach, CDS, tackles the challenge of distributional shift in data sharing by intelligently sharing data across tasks and improves multi-task performance by effectively trading off between the benefits of data sharing and the harms of excessive distributional shift.

7.2.2 *Notation and Problem Statement*

Multi-task offline RL. The goal in multi-task RL is to find a policy that maximizes expected return in a multi-task Markov decision process (MDP), defined as $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, \gamma, \{R_i, i\}_{i=1}^N)$, with state space $\mathcal{S}$, action space $\mathcal{A}$, dynamics $P(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a})$, a discount factor $\gamma \in [0, 1)$, and a finite set of task indices $1, \cdots, N$ with corresponding reward functions $R_1, \cdots, R_N$. Each task i presents a different reward function R_i, but we assume that the dynamics P are shared across tasks. While this setting is not fully general, there are a wide variety of practical problem settings for which only the reward changes including various goal navigation tasks [92], distinct object manipulation objectives [347], and different user preferences [48]. In this work, we focus on learning a policy $\pi(\boldsymbol{a}|\boldsymbol{s}, i)$, which in practice could be modelled as independent policies $\{\pi_1(\boldsymbol{a}|\boldsymbol{s}), \cdots, \pi_N(\boldsymbol{a}|\boldsymbol{s})\}$ that do not share any parameters, or as a single task-conditioned policy, $\pi(\boldsymbol{a}|\boldsymbol{s}, i)$ with parameter sharing. Our goal in this chapter is to analyze and devise methods for data sharing and the choice of parameter sharing is orthogonal, and can be made independently. We formulate the policy optimization problem as finding a policy that maximizes expected return over all the tasks: $\pi^*(\boldsymbol{a}|\boldsymbol{s}, \cdot) := \arg\max_\pi \mathbb{E}_{i\sim[N]}\mathbb{E}_{\pi(\cdot|\cdot,i)}[\sum_t \gamma^t R_i(\boldsymbol{s}_t, \boldsymbol{a}_t)]$.

Standard offline RL is concerned with learning policies $\pi(\boldsymbol{a}|\boldsymbol{s})$ using only a given static dataset of transitions $\mathcal{D} = \{(\boldsymbol{s}_j, \boldsymbol{a}_j, \boldsymbol{s}'_j, r_j)\}_{j=1}^N$, collected by a behavior policy $\pi_\beta(\boldsymbol{a}|\boldsymbol{s})$, without any additional environment interaction. In the multi-task offline RL setting, the dataset $\mathcal{D}$ is partitioned into per-task subsets, $\mathcal{D} = \cup_{i=1}^N \mathcal{D}_i$, where $\mathcal{D}_i$ consists of experience from task i. While algorithms can choose to train the policy for task i (i.e., $\pi(\cdot|\cdot, i)$) only on $\mathcal{D}_i$, in this paper, we are interested in data-sharing schemes that correspond to relabeling data from a different task, $j \neq i$ with the reward function r_i, and learn $\pi(\cdot|\cdot, i)$ on the combined data. To be able to do so, we assume access to the functional form of the reward r_i, a common assumption in goal-conditioned RL [10, 76], and which often holds in robotics applications through the use of learned classifiers [347, 161], and discriminators [89, 42].

We assume that relabeling data $\mathcal{D}_j$ from task j to task i generates a dataset $\mathcal{D}_{j\to i}$, which is then additionally used to train on task i. Thus, the effective dataset for task i after relabeling is given by $\mathcal{D}_i^{\text{eff}} := \mathcal{D}_i \cup \left(\cup_{j\neq i}\mathcal{D}_{j\to i}\right)$. This notation simply formalizes data sharing and relabeling strategies explored in prior work [76, 162]. Our aim in this paper will be to improve on this naïve strategy, which we will show leads to better results.

Offline RL algorithms. For our analysis in this chapter, we will abstract conservative offline RL algorithms into a generic constrained policy optimization problem [181]:

$$\pi^*(\boldsymbol{a}|\boldsymbol{s}) := \arg\max_{\pi} \quad J_{\mathcal{D}}(\pi) - \alpha D(\pi, \pi_\beta). \tag{7.2.1}$$

$J_{\mathcal{D}}(\pi)$ denotes the average return of policy π in the empirical MDP induced by the transitions in the dataset, and $D(\pi, \pi_\beta)$ denotes a divergence measure (e.g., KL-divergence [149, 343], MMD distance [179] or D_{CQL} [181]) between the learned policy π and the behavior policy π_β.

In the multi-task offline RL setting with data-sharing, the generic optimization problem in Equation 7.2.1 for a task i utilizes the effective dataset $\mathcal{D}_i^{\text{eff}}$. In addition, we define $\pi_\beta^{\text{eff}}(\boldsymbol{a}|\boldsymbol{s}, i)$ as the effective behavior policy for task i and it is given by: $\pi_\beta^{\text{eff}}(\boldsymbol{a}|\boldsymbol{s}, i) := |\mathcal{D}_i^{\text{eff}}(\boldsymbol{s}, \boldsymbol{a})|/|\mathcal{D}_i^{\text{eff}}(\boldsymbol{s})|$. Hence, the counterpart of Equation 7.2.1 in the multi-task offline RL setting with data sharing is given by:

$$\forall i \in [N], \quad \pi^*(\boldsymbol{a}|\boldsymbol{s}, i) := \arg\max_{\pi} \quad J_{\mathcal{D}_i^{\text{eff}}}(\pi) - \alpha D(\pi, \pi_\beta^{\text{eff}}). \tag{7.2.2}$$

We will utilize this generic optimization problem to motivate our method in Section 7.2.4.

7.2.3 *When Does Data Sharing Actually Help in Offline Multi-Task RL?*

Our goal is to leverage data from all tasks to learn a policy for a particular task of interest. Perhaps the simplest approach to leveraging experience across tasks is to train the task policy on not just the data coming from that task, but also relabeled data from all other tasks [37]. Is this naïve data sharing strategy sufficient for learning effective behaviors from multi-task offline data? In this section, we aim to answer this question via empirical analysis on a relatively simple domain, which will reveal interesting aspects of data sharing. We first describe the experimental setup and then discuss the results and possible explanations for the observed behavior. Using these insights, we will then derive a simple and effective data sharing strategy in Section 7.2.4.

Experimental analysis setup. To assess the efficacy of data sharing, we experimentally analyze various multi-task RL scenarios created with the walker2d environment in Gym [32]. We construct different test scenarios on this environment that mimic practical situations, including settings where different amounts of data of varied quality are available for different tasks [162, 348, 302]. In all these scenarios, the agent attempts three tasks: run forward, run backward, and jump, which we visualize in Figure 28. Following the problem statement in Section 7.2.2, these tasks share the same state-action space and transition dynamics, differing only in the reward function that the agent is trying to optimize. Different scenarios are generated with varying size offline datasets, each collected with policies that have different degrees of suboptimality. This might include,

Dataset types / Tasks	Dataset Size	Avg Return		$D_{\text{KL}}(\pi, \pi_\beta)$	
		No Sharing	**Sharing All**	**No Sharing**	**Sharing All**
medium-replay / run forward	109900	**998.9**	966.2	**3.70**	10.39
medium-replay / run backward	109980	**1298.6**	1147.5	**4.55**	12.70
medium-replay / jump	109511	**1603.1**	1224.7	**3.57**	15.89
average task performance	N/A	**1300.2**	1112.8	**3.94**	12.99
medium / run forward	27646	297.4	**848.7**	**6.53**	11.78
medium / run backward	31298	207.5	**600.4**	**4.44**	10.13
medium / jump	100000	351.1	**776.1**	**5.57**	21.27
average task performance	N/A	285.3	**747.7**	**5.51**	14.39
medium-replay / run forward	109900	590.1	**701.4**	**1.49**	7.76
medium / run backward	31298	614.7	**756.7**	**1.91**	12.2
expert / jump	5000	**1575.2**	885.1	**3.12**	27.5
average task performance	N/A	**926.6**	781	**2.17**	15.82

Table 11: We analyze how sharing data across all tasks (**Sharing All**) compares to **No Sharing** in the multi-task walker2d environment with three tasks: run forward, run backward, and jump. We provide three scenarios with different styles of per-task offline datasets in the leftmost column. The second column shows the number of transitions in each dataset. We report the per-task average return, the KL divergence between the single-task optimal policy π and the behavior policy π_β after the data sharing scheme, as well as averages across tasks. **Sharing All** generally helps training while increasing the KL divergence. However, on the row highlighted in yellow, **Sharing All** yields a particularly large KL divergence between the single-task π and π_β and degrades the performance, suggesting sharing data for all tasks is brittle.

for each task, a single policy with mediocre or expert performance, or a mixture of policies given by the initial part of the replay buffer trained with online SAC [126]. We refer to these three types of offline datasets as medium, expert and medium-replay, respectively, following Fu et al. [92].

We train a single-task policy $\pi_{\text{CQL}}(\boldsymbol{a}|\boldsymbol{s}, i)$ with CQL [181] as the base offline RL method, along with two forms of data-sharing, as shown in Table 11: no sharing of data across tasks (**No Sharing**) and complete sharing of data with relabeling across all tasks (**Sharing All**). In addition, we also measure the divergence term in Equation 7.2.2, $D(\pi(\cdot|\cdot, i), \pi_\beta^{\text{eff}}(\cdot|\cdot, i))$, for $\pi = \pi_{\text{CQL}}(\boldsymbol{a}|\boldsymbol{s}, i)$, averaged across tasks by using the Kullback-Liebler divergence. This value quantifies the average divergence between the single-task optimal policy and the relabeled behavior policy averaged across tasks.

Analysis of results in Table 11. To begin, note that even naïvely sharing data is better than not sharing any data at all on **5/9** tasks considered (compare the performance across **No Sharing** and **Sharing All** in Table 11). However, a closer look at Table 11 suggests that data-sharing can significantly degrade performance on certain tasks, especially in scenarios where the amount of data available for the original task is limited, and where the distribution of this data is narrow. For example, when using expert data for jumping in conjunction with more than 25 times as much lower-quality (mediocre & random)

data for running forward and backward, we find that the agent performs poorly on the jumping task despite access to near-optimal jumping data.

Why does naïve data sharing degrade performance on certain tasks despite near-optimal behavior for these tasks in the original task dataset? We argue that the primary reason that naïve data sharing can actually hurt performance in such cases is because it exacerbates the distributional shift issues that afflict offline RL. Many offline RL methods combat distribution shift by implicitly or explicitly constraining the learned policy to stay close to the training data. Then, when the training data is changed by adding relabeled data from another task, the constraint causes the learned policy to change as well. When the added data is of low quality for that task, it will correspondingly lead to a lower quality learned policy for that task, unless the constraint is somehow modified. This effect is evident from the higher divergence values between the learned policy without any data-sharing and the effective behavior policy for that task *after* relabeling (e.g., expert+jump) in Table 11. Although these results are only for CQL, we expect that any offline RL method would, insofar as it combats distributional shift by staying close to the data, would exhibit a similar problem.

To mathematically quantify the effects of data-sharing in multi-task offline RL, we appeal to safe policy improvement bounds [194, 181, 367] and discuss cases where data-sharing between tasks i and j can degrade the amount of worst-case guaranteed improvement over the behavior policy. Prior work [181] has shown that the generic offline RL objective in Theorem 5.1.5 enjoys the following guarantees of policy improvement on the actual MDP, beyond the behavior policy:

$$J(\pi^*) \geq J(\pi_\beta) - \mathcal{O}(1/(1-\gamma)^2)\mathbb{E}_{\boldsymbol{s},\boldsymbol{a}\sim d^\pi}\left[\sqrt{\frac{D(\pi(\cdot|\boldsymbol{s}),\pi_\beta(\cdot|\boldsymbol{s}))}{|\mathcal{D}(\boldsymbol{s})|}}\right] + \alpha/(1-\gamma)D(\pi,\pi_\beta). \tag{7.2.3}$$

We will use Equation 7.2.3 to understand the scenarios where data sharing can hurt. When data sharing modifies $\mathcal{D} = \mathcal{D}_i$ to $\mathcal{D} = \mathcal{D}_i^{\text{eff}}$, which includes $\mathcal{D}_i$ as a subset, it effectively aims at reducing the magnitude of the second term (i.e., sampling error) by increasing the denominator. This can be highly effective if the state distribution of the learned policy π^* and the dataset $\mathcal{D}$ overlap. However, an increase in the divergence $D(\pi(\cdot|\boldsymbol{s}), \pi^\beta(\cdot|\boldsymbol{s}))$ as a consequence of relabeling implies a potential increase in the sampling error, unless the increased value of $|\mathcal{D}^{\text{eff}}(\boldsymbol{s})|$ compensates for this. Additionally, the bound also depends on the quality of the behavior data added after relabeling: if the resulting behavior policy π_β^{eff} is more suboptimal compared to π_β, i.e., $J(\pi_\beta^{\text{eff}}) < J(\pi_\beta)$, then the guaranteed amount of improvement also reduces.

To conclude, our analysis reveals that while data sharing is often helpful in multi-task offline RL, it can lead to substantially poor performance on certain tasks as a result of

exacerbated distributional shift between the optimal policy and the effective behavior policy induced after sharing data.

7.2.4 *CDS: Reducing Distributional Shift in Multi-Task Data Sharing*

The analysis in Section 7.2.3 shows that naïve data sharing may be highly sub-optimal in some cases, and although it often does improve over no data sharing at all, it can also lead to exceedingly poor performance. Can we devise a conservative approach that shares data intelligently to not exacerbate distributional shift as a result of relabeling?

7.2.4.1 *A First Attempt at Designing a Data Sharing Strategy*

A straightforward data sharing strategy is to utilize a transition for training only if it reduces the distributional shift. Formally, this means that for a given transition $(\boldsymbol{s}, \boldsymbol{a}, r_j(\boldsymbol{s}, \boldsymbol{a}), \boldsymbol{s}') \in \mathcal{D}_j$ sampled from the dataset $\mathcal{D}_j$, such a scheme would prescribe using it for training task i (i.e., $(\boldsymbol{s}, \boldsymbol{a}, r_i(\boldsymbol{s}, \boldsymbol{a}), \boldsymbol{s}') \in \mathcal{D}_i^{\text{eff}}$) only if:

$$\textbf{CDS (basic)}: \quad \Delta^{\pi}(\boldsymbol{s}, \boldsymbol{a}) := D(\pi(\cdot|\cdot, i), \pi_\beta(\cdot|\cdot, i))(\boldsymbol{s}) - D(\pi(\cdot|\cdot, i), \pi_\beta^{\text{eff}}(\cdot|\cdot, i))(\boldsymbol{s}) \geq 0. \tag{7.2.4}$$

The scheme presented in Equation 7.2.4 would guarantee that distributional shift (i.e., second term in Equation 7.2.2) is reduced. Moreover, since sharing data can only increase the size of the dataset and not reduce it, this scheme is guaranteed to not increase the sampling error term in Equation 7.2.3. We refer to this scheme as the basic variant of conservative data sharing (**CDS (basic)**).

While this scheme can prevent the negative effects of increased distributional shift, this scheme is quite pessimistic. Even in our experiments, we find that this variant of CDS does not improve performance by a large margin. Additionally, as observed in Table 11 (medium-medium-medium data composition) and discussed in Section 7.2.3, data sharing can often be useful despite an increased distributional shift (note the higher values of $D_{\text{KL}}(\pi, \pi_\beta)$ in Table 11) likely because it reduces sampling error and potentially utilizes data of higher quality for training. **CDS (basic)** described above does not take into account these factors. Formally, the effect of the first term in Equation 7.2.2, $J_{\mathcal{D}^{\text{eff}}}(\pi)$ (the policy return in the empirical MDP generated by the dataset) and a larger increase in $|\mathcal{D}^{\text{eff}}(\boldsymbol{s})|$ at the cost of somewhat increased value of $D(\pi(\cdot|\boldsymbol{s}), \pi_\beta(\cdot|\boldsymbol{s}))$ are not taken into account. Thus we ask: can we instead design a more complete version of CDS that effectively balances the tradeoff by incorporating all the discussed factors (distributional shift, sampling error, data quality)?

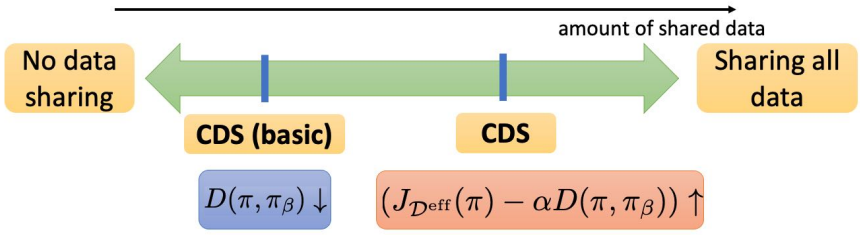


Figure 27: A schematic comparing **CDS** and **CDS (basic)** data sharing schemes relative to no sharing (left extreme) and full data sharing (right extreme). While p-CDS only shares data when distributional shift is strictly reduced, o-CDS is more optimistic and shares data when the objective in Equation 7.2.2 is larger. Typically, we would expect that CDS shares more transitions than CDS (basic).

7.2.4.2 *The Complete Version of Conservative Data Sharing (CDS)*

Next, we present the complete version of our method. The complete version of CDS, which we will refer to as **CDS**, for notational brevity is derived from the following perspective: we note that a data sharing scheme can be viewed as altering the dataset $\mathcal{D}_i^{\text{eff}}$, and hence the effective behavior policy, $\pi_\beta^{\text{eff}}(\boldsymbol{a}|\boldsymbol{s}, i)$. Thus, we can directly *optimize* the objective in Equation 7.2.2 with respect to π_β^{eff}, in addition to π, where π_β^{eff} belongs to the set of all possible effective behavior policies that can be obtained via any form of data sharing. Note that unlike CDS (basic), this approach would not rely on only indirectly controlling the objective in Equation 7.2.2 by controlling distributional shift, but would aim to directly optimize the objective in Equation 7.2.2. We formalize this as:

$$\arg\max_{\pi} \max_{\pi_\beta^{\text{eff}} \in \Pi_{\text{relabel}}} \left[J_{\mathcal{D}_i^{\text{eff}}}(\pi) - \alpha D(\pi, \pi_\beta^{\text{eff}}; i) \right], \tag{7.2.5}$$

where Π_{relabel} denotes the set of all possible behavior policies that can be obtained via relabeling. The next result characterizes safe policy improvement for Equation 7.2.5 and discusses how it leads to improvement over the behavior policy.

Proposition 7.2.1 (Characterizing safe-policy improvement for CDS.). *Let $\pi^*(\boldsymbol{a}|\boldsymbol{s})$ be the policy obtained from Equation 7.2.5, and let $\pi_\beta(\boldsymbol{a}|\boldsymbol{s})$ be the behavior policy for $\mathcal{D}_i$. Then, w.h.p. $\geq 1 - \delta$, π^* is a ζ-safe policy improvement over π_β, i.e., $J(\pi^*) \geq J(\pi_\beta) - \zeta$, where ζ is given by:*

$$\zeta = \mathcal{O}\left(\frac{1}{(1-\gamma)^2}\right) \mathbb{E}_{\boldsymbol{s} \sim d^{\pi^*}_{\mathcal{D}_i^{\text{eff}}}} \left[\sqrt{\frac{D_{CQL}(\pi^*, \pi^*_\beta)(\boldsymbol{s}) + 1}{|\mathcal{D}_i^{\text{eff}}(\boldsymbol{s})|}} \right] - \left[\alpha D(\pi^*, \pi^*_\beta) + \underbrace{J(\pi^*_\beta) - J(\pi_\beta)}_{(a)} \right],$$

where $\mathcal{D}_i^{\text{eff}} \sim d^{\pi^*_\beta}(\boldsymbol{s})$ and $\pi^*_\beta(\boldsymbol{a}|\boldsymbol{s})$ denotes the policy $\pi \in \Pi_{relabel}$ that maximizes Equation 7.2.5.

A proof and analysis of this proposition is provided in Appendix E.2, where we note that the bound in Proposition 7.2.1 is stronger than both no data sharing as well as naïve data sharing. We show in Appendix E.2 that optimizing Equation 7.2.5 reduces the numerator $D_{\text{CQL}}(\pi^*, \pi^*_\beta)$ term while also increasing $|\mathcal{D}^{\text{eff}}_i(\boldsymbol{s})|$, thus reducing the amount of sampling error. In addition, Lemma E.2.2 shows that the improvement term (a) is guaranteed to be positive if a large enough α is chosen in Equation 7.2.5. Combining these, we find data sharing using Equation 7.2.5 improves over both complete data sharing (which may increase $D_{\text{CQL}}(\pi, \pi_\beta)$) and no data sharing (which does not increase $|\mathcal{D}^{\text{eff}}_i(\boldsymbol{s})|$). A schematic comparing the two variants of CDS and naïve and no data sharing schemes is shown in Figure 27.

Optimizing Equation 7.2.5 tractably. The next step is to effectively convert Equation 7.2.5 into a simple condition for data sharing in multi-task offline RL. While directly solving Equation 7.2.5 is intractable in practice, since both the terms depend on $\pi^{\text{eff}}_\beta(\boldsymbol{a}|\boldsymbol{s})$ (since the first term $J_{\mathcal{D}^{\text{eff}}}(\pi)$ depends on the empirical MDP induced by the effective behavior policy and the amount of sampling error), we need to instead solve Equation 7.2.5 approximately. Fortunately, we can optimize a *lower-bound approximation* to Equation 7.2.5 that uses the dataset state distribution for the policy update in Equation 7.2.5 similar to modern actor-critic methods [58, 211, 97, 126, 181] which only introduces an additional $D(\pi, \pi_\beta)$ term in the objective. This objective is given by: $\mathbb{E}_{\boldsymbol{s}\sim\mathcal{D}^{\text{eff}}_i}[\mathbb{E}_\pi[Q(\boldsymbol{s}, \boldsymbol{a}, i)] - \alpha' D(\pi(\cdot|\boldsymbol{s}, i), \pi^{\text{eff}}_\beta(\cdot|\boldsymbol{s}, i))]$, which is equal to the expected "conservative Q-value" $\hat{Q}^\pi(\boldsymbol{s}, \boldsymbol{a}, i)$ on dataset states, policy actions and task i. Optimizing this objective via a co-ordinate descent on π and π^{eff}_β dictates that π be updated using a standard update of maximizing the conservative Q-function, $\hat{Q}^\pi$ (equal to the difference of the Q-function and $D(\pi, \pi^{\text{eff}}_\beta; i)$). Moreover, π^{eff}_β should also be updated towards maximizing the same expectation, $\mathbb{E}_{\boldsymbol{s},\boldsymbol{a}\sim\mathcal{D}^{\text{eff}}_i}[\hat{Q}^\pi(\boldsymbol{s}, \boldsymbol{a}, i)] := \mathbb{E}_{\boldsymbol{s},\boldsymbol{a}\sim\mathcal{D}^{\text{eff}}_i}[Q(\boldsymbol{s}, \boldsymbol{a}, i)] - \alpha D(\pi, \pi^{\text{eff}}_\beta; i)$. This implies that when updating the behavior policy, we should prefer state-action pairs that maximize the conservative Q-function.

Deriving the data sharing strategy for CDS. Utilizing the insights for optimizing Equation 7.2.5 tractably, we now present the effective data sharing rule prescribed by CDS. For any given task i, we want relabeling to incorporate transitions with the highest conservative Q-value into the resulting dataset $\mathcal{D}^{\text{eff}}_i$, as this will directly optimize the tractable lower bound on Equation 7.2.5. While directly optimizing Equation 7.2.5 will enjoy benefits of reduced sampling error since $J_{\mathcal{D}^{\text{eff}}_i}(\pi)$ also depends on sampling error, our tractable lower bound approximation does not enjoy this benefit. This is because optimizing the lower-bound only increases the frequency of a state in the dataset, $|\mathcal{D}^{\text{eff}}_i(\boldsymbol{s})|$ by atmost 1. To encourage further reduction in sampling error, we modify CDS to instead share all transitions with a conservative Q-value more than the top k^{th} quantile of the original dataset $\mathcal{D}_i$, where k is a hyperparameter. This provably increases the objective

value in Equation 7.2.5 still ensuring that term $(a) > 0$ in Proposition 7.2.1, while also reducing $|\mathcal{D}_i^{\text{eff}}(\boldsymbol{s})|$ in the denominator. Thus, for a given transition $(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}') \in \mathcal{D}_j$,

$$\textbf{CDS:} \quad (\boldsymbol{s}, \boldsymbol{a}, r_i, \boldsymbol{s}') \in \mathcal{D}_i^{\text{eff}} \text{ if } \Delta^\pi(\boldsymbol{s}, \boldsymbol{a}) := \hat{Q}^\pi(\boldsymbol{s}, \boldsymbol{a}, i) - P_{k\%}\{\hat{Q}^\pi(\boldsymbol{s}', \boldsymbol{a}', i) \colon \boldsymbol{s}', \boldsymbol{a}' \sim \mathcal{D}_i\} \geq 0, \tag{7.2.6}$$

where $\hat{Q}^\pi$ denotes the learned conservative Q-function estimate. If the condition in Equation 7.2.6 holds for the given $(\boldsymbol{s}, \boldsymbol{a})$, then the corresponding relabeled transition, $(\boldsymbol{s}, \boldsymbol{a}, r_i(\boldsymbol{s}, \boldsymbol{a}), \boldsymbol{s}')$ is added to $\mathcal{D}_i^{\text{eff}}$.

We summarize the pesudocode of CDS in Algorithm 6 in Appendix E.1 and include the practical implementation details of CDS in Appendix E.3.

7.2.5 *Experimental Evaluation of Conservative Data Sharing*

We conduct experiments to answer six main questions: **(1)** can CDS prevent performance degradation when sharing data as observed in Section 7.2.3?, **(2)** how does CDS compare to vanilla multi-task offline RL methods and prior data sharing methods? **(3)** can CDS handle sparse reward settings, where data sharing is particularly important due to scarce supervision signal? **(4)** can CDS handle goal-conditioned offline RL settings where the offline dataset is undirected and highly suboptimal? **(5)** Can CDS scale to complex visual observations? **(6)** Can CDS be combined with any offline RL algorithms? Besides these questions, we visualize CDS weights for better interpretation of the data sharing scheme learned by CDS in Figure 37 in Appendix E.4.2.

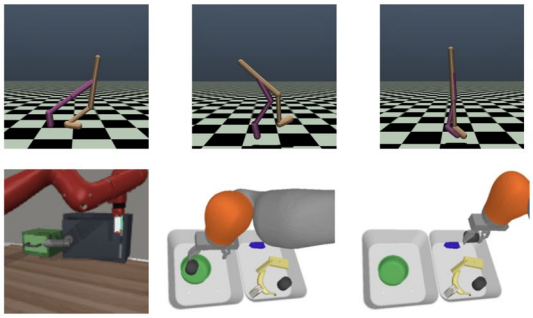


Figure 28: Environments (from left to right): walker2d run forward, walker2d run backward, walker2d jump, Meta-World door open/close and drawer open/close and vision-based pick-place tasks in [162].

Comparisons. To answer these questions, we consider the following prior methods. On tasks with low dimensional state spaces, we compare with the online multi-task relabeling approach **HIPI** [76], which uses inverse RL to infer for which tasks the datapoints are optimal and in practice routes a transition to task with the highest Q-value. We adapt HIPI to the offline setting by applying its data routing strategy to a conservative offline RL algorithm. We also compare to naïvely sharing data across all tasks (denoted as **Sharing All**) and vanilla multi-task offline RL method without any data sharing (denoted as **No Sharing**). On image-based domains, we compare CDS to the data sharing strategy based on human-defined skills [162] (denoted as **Skill**), which manually groups tasks into different skills (e.g. skill "pick" and skill

"place") and only routes an episode to target tasks that belongs to the same skill. In these domains, we also compare to **HIPI**, **Sharing All** and **No Sharing**. Beyond these multi-task RL approaches with data sharing, to assess the importance of data sharing in offline RL, we perform an additional comparison to other alternatives to data sharing in multi-task offline RL settings. One traditionally considered approach is to use data from other tasks for some form of "pre-training" before learning to solve the actual task. We instantiate this idea by considering a method from Yang and Nachum [354] that conducts contrastive representation learning on the multi-task datasets to extract shared representation between tasks and then runs multi-task RL on the learned representations. We discuss this comparison in detail in Table 51 in Appendix E.4.3. To answer question (6), we use CQL [181] (a Q-function regularization method) and BRAC [343] (a policy-constraint method) as the base offline RL algorithms for all methods. We discuss evaluations of CDS with CQL in the main text and include the results of CDS with BRAC in Table 49 in Appendix E.4.1. For more details on setup and hyperparameters, see Appendix E.3.

Multi-task environments. We consider a number of multi-task reinforcement learning problems on environments visualized in Figure 28. To answer questions (1) and (2), we consider the walker2d locomotion environment from OpenAI Gym [32] with dense rewards. We use three tasks, run forward, run backward and jump, as proposed in prior offline RL work [365]. To answer question (3), we also evaluate on robotic manipulation domains using environments from the Meta-World benchmark [364]. We consider four tasks: door open, door close, drawer open and drawer close. Meaningful data sharing requires a consistent state representation across tasks, so we put both the door and the drawer on the same table, as shown in Figure 28. Each task has a sparse reward of 1 when the success condition is met and 0 otherwise. To answer question (4), we consider maze navigation tasks where the temporal "stitching" ability of an offline RL algorithm is crucial to obtain good performance. We create goal reaching tasks using the ant robot in the medium and hard mazes from D4RL [92]. The set of goals is a fixed discrete set of size 7 and 3 for large and medium mazes, respectively. Following Fu et al. [92], a reward of +1 is given and the episode terminates if the state is within a threshold radius of the goal. Finally, to explore how CDS scales to image-based manipulation tasks (question (5)), we utilize a simulation environment similar to the real-world setup presented in [162]. This environment, which was utilized by Kalashnikov et al. [162] as a representative and realistic simulation of a real-world robotic manipulation problem, consists of 10 image-based manipulation tasks that involve different combinations of picking specific objects (banana, bottle, sausage, milk box, food box, can and carrot) and placing them in one of the three fixtures (bowl, plate and divider plate) (see example task images in Fig. 28). More environment details are in the appendix. We report the average return for locomotion tasks and success rate for AntMaze and both manipluation environments,

averaged over 6 and 3 random seeds for environments with low-dimensional inputs and image inputs respectively.

Environment	Dataset types / Tasks	$D_{\mathrm{KL}}(\pi, \pi_\beta)$ No Sharing	Sharing All	CDS (basic) (ours)	CDS (ours)
walker2d	medium-replay / run forward	**1.49**	7.76	14.31	**1.49**
	medium / run backward	**1.91**	12.2	8.26	6.09
	expert / jump	3.12	27.5	13.25	**2.91**

Table 12: Measuring $D_{\mathrm{KL}}(\pi, \pi_\beta)$ on the walker2d environment. **Sharing All** degrades the performance on task jump with limited expert data as discussed in Table 11. CDS manages to obtain a π_β after data sharing that is closer to the single-task optimal policy in terms of the KL divergence compared to **No Sharing** and **Sharing All** on task jump (highlighted in yellow). Since CDS also achieves better performance, this analysis suggests that reducing distribution shift is important for effective offline data sharing.

Multi-task datasets. Following the analysis in Section 7.2.3, we intentionally construct datasets with a variety of heterogeneous behavior policies to test if CDS can provide effective data sharing to improve performance while avoiding harmful data sharing that exacerbates distributional shift. For the locomotion domain, we use a large, diverse dataset (medium-replay) for run forward, a medium-sized dataset for run backward, and an expert dataset with limited data for run jump. For Meta-World, we consider medium-replay datasets with 152K transitions for task door close and drawer open and expert datasets with only 2K transitions for task door open and drawer close. For AntMaze, we modify the D4RL datasets for antmaze-*-play environments to construct two kinds of multi-task datasets: an "undirected" dataset, where data is equally divided between different tasks and the rewards are correspondingly relabeled, and a "directed" dataset, where a trajectory is associated with the goal closest to the final state of the trajectory. This means that the per-task data in the undirected setting may not be relevant to reaching the goal of interest. Thus, data-sharing is crucial for good performance: methods that do not effectively perform data sharing and train on largely task-irrelevant data are expected to perform worse. Finally, for image-based manipulation tasks, we collect datasets for all the tasks individually by running online RL [161] until the task reaches medium-level performance (40% for picking tasks and 80% placing tasks). At that point, we merge the entire replay buffers from different tasks creating a final dataset of 100K RL episodes with 25 transitions for each episode.

Results on domains with low-dimensional states. We present the results on all non-vision environments in Table 18. CDS achieves the best average performance across all environments except that on walker2d, it achieves the second best performance, obtaining slightly worse the other variant CDS (basic). On the locomotion domain, we observe the most significant improvement on task jump on all three environments. We interpret this as strength of conservative data sharing, which mitigates the distribution shift that can be introduced by routing large amount of other task data to the task with limited data and narrow distribution. We also validate this by measuring the $D_{\mathrm{KL}}(\pi, \pi_\beta)$ in

Environment	Tasks / Dataset type	CDS (ours)	CDS (basic)	HIPI [76]	Sharing All	No Sharing
walker2d	run forward / medium-replay	**1057.9**±121.6	968.6±188.6	695.5±61.9	701.4±47.0	590.1±48.6
	run backward / medium	564.8±47.7	594.5±22.7	626.0±48.0	**756.7**±76.7	614.7±87.3
	jump / expert	1418.2±138.4	1501.8±115.1	**1603.7**±146.8	885.1±152.9	1575.2±70.9
	average	1013.6±71.5	**1021.6**±76.9	975.1±45.1	781.0±100.8	926.6±37.7
Meta-World [364]	door open / expert	**58.4%**±9.3%	30.1%±16.6%	26.5%±20.5%	34.3%±17.9%	14.5%±12.7
	door close / medium-replay	**65.3%**±27.7%	41.5%±28.2%	1.3%±5.3%	48.3%±27.3%	4.0%±6.1%
	drawer open / medium-replay	**57.9%**±16.2%	39.4%±16.9%	41.2%±24.9%	55.1%±9.4%	16.0%±17.5%
	drawer close / expert	98.8%±0.7%	86.3%±0.9%	62.2%±33.4%	**100.0%**±0%	99.0%±0.7%
	average	**70.1%**±8.1%	49.3%±16.0%	32.8%±18.7%	59.4%±5.7%	33.4%±8.3%
AntMaze [92]	large maze (7 tasks) / undirected	**22.8**% ± 4.5%	10.0% ± 5.9%	1.3% ± 2.3%	16.7% ± 7.0%	13.3% ± 8.6%
	large maze (7 tasks) / directed	**24.6%** ± 4.7%	0.0% ± 0.0%	11.8% ± 5.4%	20.6% ± 4.4%	19.2% ± 8.0%
	medium maze (3 tasks) / undirected	**36.7%** ± 6.2%	0.0% ± 0.0%	8.6% ± 3.2%	22.9% ± 3.6%	21.6% ± 7.1%
	medium maze (3 tasks) / directed	**18.5**% ± 6.0%	0.0% ± 0.0%	8.3% ± 9.1%	12.4% ± 5.4%	**17.0%** ± 3.2%

Table 13: Results for multi-task locomotion (walker2d), robotic manipulation (Meta-World) and navigation environments (AntMaze) with low-dimensional state inputs. Numbers are averaged across 6 seeds, ± the 95%-confidence interval. We include per-task performance for walker2d and Meta-World domains and the overall performance averaged across tasks (highlighted in gray) for all three domains. We bold the highest score across all methods. CDS achieves the best or comparable performance on all of these environments.

Task Name	CDS (ours)	HIPI [76]	Skill [162]	Sharing All	No Sharing
lift-banana	**53.1%**±3.2%	48.3%±6.0%	32.1%±9.5%	41.8%±4.2%	20.0%±6.0%
lift-bottle	**74.0%**±6.3%	64.4%±7.7%	55.9%±9.6%	60.1%±10.2%	49.7%±8.7%
lift-sausage	**71.8%**±3.9%	71.0%±7.7%	68.8%±9.3%	70.0%±7.0%	60.9%±6.6%
lift-milk	**83.4%**±5.2%	79.0%±3.9%	68.2%±3.5%	72.5%±5.3%	68.4%±6.1%
lift-food	61.4%±9.5%	**62.6%**±6.3%	41.5%±12.1%	58.5%±7.0%	39.1%±7.0%
lift-can	65.5%±6.9%	**67.8%**±6.8%	50.8%±12.5%	57.7%±7.2%	49.1%±9.8%
lift-carrot	**83.8%**±3.5%	78.8%±6.9%	66.0%±7.0%	75.2%±7.6%	69.4%±7.6%
place-bowl	**81.0%**±8.1%	77.2%±8.9%	80.8%±6.9%	70.8%±7.8%	80.3%±8.6%
place-plate	85.8%±6.6%	83.6%±7.9%	78.4%±9.6%	78.7%±7.6%	**86.1**%±7.7%
place-divider-plate	**87.8%**±7.6%	78.0%±10.5%	80.8%±5.3%	79.2%±6.3%	85.0%±5.9%
average	**74.8%**±6.4%	71.1%±7.5%	62.3%±8.9%	66.4%±7.2%	60.8%±7.5%

Table 14: Results for multi-task vision-based robotic manipulation from [162]. Numbers are averaged across 3 seeds, ± the 95% confidence interval. We consider 7 tasks denoted as lift-object where the goal of each task is to lift a different object and 3 tasks denoted as place-fixture that aim to place a lifted object onto different fixtures. CDS outperforms both a skill-based data sharing strategy [162] (**Skill**) and other data sharing methods on the average success rate (highlighted in gray) and 7 out of 10 per-task success rates.

Table 12 where π_β is the behavior policy after we perform CDS to share data. As shown in Table 12, CDS achieves lower KL divergence between the single-task optimal policy and the behavior policy after data sharing on task jump with limited expert data, whereas **Sharing All** results in much higher KL divergence compared to **No Sharing** as discussed in Section 7.2.3 and Table 11. Hence, CDS is able to mitigate distribution shift when sharing data and result in performance boost.

On the Meta-World tasks, we find that the agent without data sharing completely fails to solve most of the tasks due to the low quality of the medium replay datasets and

the insufficient data for the expert datasets. **Sharing All** improves performance since in the sparse reward settings, data sharing can introduce more supervision signal and help training. CDS further improves over **Sharing All**, suggesting that CDS can not only prevent harmful data sharing, but also lead to more effective multi-task learning compared to **Sharing All** in scenarios where data sharing is imperative. It's worth noting that CDS (basic) performs worse than CDS and **Sharing All**, indicating that relabeling data that only mitigates distributional shift is too pessimistic and might not be sufficient to discover the shared structure across tasks.

On AntMaze, we observe that CDS performs better than **Sharing All** and significantly outperforms HIPI in all four settings. Perhaps surprisingly, **No Sharing** is a strong baseline, however, is outperformed by CDS with the harder undirected data. Moreover, CDS performs on-par or better in the undirected setting compared to the directed setting, indicating the effectiveness of CDS in routing data in challenging settings.

Results on image-based robotic manipulation domains. Here, we compare CDS to the hand-designed **Skill** sharing strategy, in addition to the other methods. Given that CDS achieves significantly better performance than CDS (basic) on low-dimensional robotic manipulation tasks in Meta-World, we only evaluate CDS in the vision-based robotic manipulation domains. Since CDS is applicable to any offline multi-task RL algorithm, we employ it as a separate data-sharing strategy in [162] while keeping the model architecture and all the other hyperparameters constant, which allows us to carefully evaluate the influence of data sharing in isolation. The results are reported in Table 19. CDS outperforms both **Skill** and other approaches, indicating that CDS is able to scale to high-dimensional observation inputs and can effectively remove the need for manual curation of data sharing strategies.

7.3 MULTI-TASK OFFLINE RL WITH UNLABELED DATA

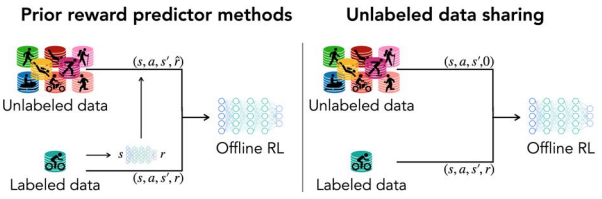


Figure 29: Overview of various methods dealing with unlabeled offline data. UDS is a simple method compared to the complex reward learning approaches yet achieves effective results.

7.3.1 *Related Work*

RL with unlabeled data. Prior works tackle the problem of learning from data without reward labels via either directly imitating expert trajectories [264, 279, 139], learning reward functions from expert data using inverse RL [1, 247, 377, 84, 88, 172], or learning a reward / value classifier that discriminates successes and failures [347, 89, 301, 77] in

standard online RL settings. Unlike these prior works, the goal of our paper is not to propose a sophisticated new algorithm for reward learning, but rather to study when the simple solution of using zero as the reward label can work well from offline data. While Singh et al. [303] also considers relabeling prior data with zero reward in the standard, single-task offline RL setting, the key distinction is that the unlabeled data in Singh et al. [303] cannot solve the target task and *actually* gets zero rewards and is thus labeled with the true reward. Unlike Singh et al. [303], we show that UDS can surprisingly also be effective even when the unlabeled data is incorrectly labeled with zero reward, and discuss how it can be improved (Section 7.3.2.2). Additionally, we consider both single-task and multi-task offline RL settings.

Data sharing. As discussed in Section 7.2.4, prior works share data based on learned Q-values [76, 205] (including our CDS approach), domain knowledge [162], and other structural quantities such as distance to goals in goal-conditioned settings [10, 216, 309, 214, 41], or the learned distance with robust inference in the offline meta-RL setting [207]. However, all of these either require access to the functional form of the reward for relabling or are limited to goal-conditioned settings. In contrast, we attempt to tackle a more general problem where reward labels are not provided for a subset of the data, and find that a simple approach for relabeling data from other tasks with the constant zero (or the minimal possible reward) can work well.

7.3.2 *How To Use Unlabeled Data in Offline RL*

Arguably the easiest approach to utilize unlabeled diverse data in offline RL is to simply assign the lowest possible reward to all the transitions in the unlabeled data, which we will assume to be 0 without loss of generality, and use this data for training the underlying (multi-task) offline RL method. We will refer to this approach as *unlabeled data sharing*, or UDS, in short. We will show that, although this strategy might seem simplistic, in fact, it can work well both in theory and practice, when the dataset composition and the task satisfies certain criteria. Not all tasks and datasets satisfy these criteria, but we will argue that many realistic offline RL problems do satisfy them. For example, UDS can work well in a problem where the labeled data consists of high-quality, near-expert demonstrations, while the unlabeled data consists of mediocre or low-reward interactions in the environment, as is often the case in robotics [348]. We will analyze the performance of this approach theoretically in Section 7.3.2.1 and empirically in Section 7.3.4.

Although the simple UDS approach can perform well in some scenarios, relabeling with zero reward of course also biases the learning process. We therefore further show that reward bias can be mitigated if the unlabeled transitions are additionally reweighted, so as to change the unlabeled data distribution (while still using a label of zero for all unlabeled data). While such reweighting reduces the sample size, it can provide an

overall benefit by reducing the reward bias and distributional shift. We will derive the optimal scheme for reweighting the transitions in the unlabeled dataset that minimizes the impact of reward bias in Section 7.3.2.2. Surprisingly, our analysis reveals that a near-optimal solution to reducing reward bias is to combine UDS with an already existing reweighting scheme for multi-task offline RL with full reward information. For example, one can choose to reweight unlabeled data with the CDS scheme of Yu et al. [366], which preferentially upweights transitions based on their conservative Q-values.

7.3.2.1 *Theoretical Analysis of UDS in Offline RL*

In this section, we will derive a performance bounds for UDS that allow us to understand several cases when this simple approach still works well. These bounds are provided in the context of the single-task offline RL problem for compactness and ease of understanding, though these can be combined together with the analysis of CDS to provide guarantees for multi-task offline RL with unlabeled data. We will show that the increase in the effective dataset size can often outweigh the bias incurred from using the wrong reward in several practical situations. We will also discuss conditions on the data composition and the relative distributions of labeled and unlabeled data that enable UDS to be successful. Formally, UDS trains on an effective dataset given by: $\mathcal{D}^{\text{eff}} = \mathcal{D}_{\text{L}} \cup \{(\boldsymbol{s}_j, \boldsymbol{a}_j, \boldsymbol{s}'_j, 0) \in \mathcal{D}_{\text{U}}\}$. We now present a policy improvement guarantee for UDS below, and then analyze the bound under special conditions. Our theoretical result builds on techniques for showing safe policy improvement bounds [194, 181, 366].

Theorem 7.3.1 (Policy improvement guarantee for UDS). *Let π^*_{UDS} denote the policy learned by UDS, and let $\pi^{\text{eff}}_\beta(\boldsymbol{a}|\boldsymbol{s})$ denote the behavior policy for the combined dataset $\mathcal{D}^{\text{eff}}$. Then with high probability $\geq 1-\delta$, π^*_{UDS} is a safe policy improvement over π^{eff}_β, i.e.,*

$$J(\pi^*_{UDS}) \geq J(\pi^{\text{eff}}_\beta) - \zeta_{err} + \underbrace{\frac{\alpha}{1-\gamma} D(\pi^*_{UDS}, \pi^{\text{eff}}_\beta)}_{\text{(c): policy improvement}},$$

$$\zeta_{err} = \underbrace{\text{RewardBias}(\pi^*_{UDS}, \pi^{\text{eff}}_\beta)}_{(a)} + \underbrace{\mathcal{O}\left(\frac{\gamma}{(1-\gamma)^2}\right) \mathbb{E}_{\boldsymbol{s},\boldsymbol{a}\sim\widehat{d}^{\pi}} \left[\sqrt{\frac{D_{CQL}(\pi^*_{UDS}, \pi^{\text{eff}}_\beta)(\boldsymbol{s})}{|\mathcal{D}^{\text{eff}}(\boldsymbol{s})|}}\right]}_{\text{(b): sampling error}},$$

$$(a) := \frac{\sum_{\boldsymbol{s},\boldsymbol{a}} \Delta\left(\widehat{d}^{\pi_\beta{}^{\text{eff}}}, \widehat{d}^{\pi^*_{UDS}}\right) \cdot (1 - f(\boldsymbol{s},\boldsymbol{a})) \cdot r(\boldsymbol{s},\boldsymbol{a})}{1-\gamma},$$

where $f(\boldsymbol{s},\boldsymbol{a}) := \frac{|\mathcal{D}_{\text{L}}(\boldsymbol{s},\boldsymbol{a})|}{|\mathcal{D}^{\text{eff}}(\boldsymbol{s},\boldsymbol{a})|}$ and $\Delta(\widehat{d}^{\pi_\beta{}^{\text{eff}}}, \widehat{d}^{\pi^*_{UDS}}) = \widehat{d}^{\pi_\beta{}^{\text{eff}}}(\boldsymbol{s},\boldsymbol{a}) - \widehat{d}^{\pi^*_{UDS}}(\boldsymbol{s},\boldsymbol{a})$.

Table 15: Summary of scenarios where UDS is expected to work and where it is not expected to work. **L** denotes the characteristics of labeled data, **U** denotes characteristics of unlabeled data. Limited/Abundant refers to the relative amount of data available High-quality/medium-quality/low-quality refers to the actual performance of the behavior policy generating the datasets. Narrow/broad refers to the relative state coverage of the datasets that we study where high coverage can lead to low distributional shift during offline RL. We provide intuitions on why UDS works or does not work under each scenario and refer to the cases shown in our analysis in Section 7.3.2.1.

Scenarios	UDS	Intuition
L: limited + high-quality + narrow, **U**: abundant + low/medium-quality + broad	✓	increase coverage (case **2** and **3**)
L: limited + medium-quality + narrow, **U**: abundant + low/medium-quality + broad	×	reward bias outweighs high coverage
L: limited + high-quality + narrow, **U**: abundant + high-quality + narrow	✓	identical distribution (case **1**)
L: limited + low-quality + narrow, **U**: abundant + high-quality + narrow	✓	increase data quality (case **2**)

A proof for Theorem 7.3.1 is provided in Appendix E.5.1. Intuitively, term (a) corresponds to the potential suboptimality incurred as a result of using the wrong reward, term (b) corresponds to the suboptimality induced due to sampling error. Finally, term (c) corresponds to the policy improvement in the empirical MDP induced by the transitions in $\mathcal{D}^{\text{eff}}$ that occurs as a result of offline RL. Intuitively, we expect that including more unlabeled data will reduce the sampling error (b) and potentially increase how much we can improve the policy (c), while potentially increasing the reward bias term (a). The key question is under which conditions we would expect the increase in bias (a) to be lower than the decrease in term (b) obtained from using the unlabeled data. We examine this below, presenting a few common cases where we expect this tradeoff to be favorable.

(1) Unlabeled data is distributed identically as labeled data. The first case we consider is when the distribution of state-action pairs in $\mathcal{D}_{\text{L}}$ is identical to the distribution of state-action pairs in the unlabeled dataset $\mathcal{D}_U$. This can arise in many application domains, such as in robotics [348, 53], where a large amount of offline data is available, but only a limited uniformly-selected fraction of it can be annotated with rewards. In this case, the fraction $f(\boldsymbol{s}, \boldsymbol{a})$ takes on identical values for all state-action pairs, and term (a) simply reverts to be a difference between the performance of the effective behavior policy and the learned policy, in the empirical MDP. Since offline RL algorithms would improve over the effective behavior policy in the empirical MDP, this term is negative and hence, no additional suboptimality is incurred in the reward bias term. Moreover, the sampling error term (b) reduces when more data is utilized. Thus, in this case, UDS improves performance due to an increase in the dataset size $|\mathcal{D}^{\text{eff}}(\boldsymbol{s})|$, without incurring a cost due to the wrong reward.

(2) Quality of the unlabeled dataset. In practice, we are often provided with a small amount of high-quality reward annotated demonstration data, and a lot of unlabeled prior data of low or mediocre quality. In this case, assigning a low reward to the transitions in the unlabeled dataset does not negatively affect policy performance significantly because the bias due to wrong reward is low (due to low-quality of labeled data) and reduces

sampling error (term (b)). On the other hand, when the unlabeled data is of high quality and large in size compared to the labeled dataset, even then the performance of the policy $J(\pi^*_{\text{UDS}})$ can be improved by using this unlabeled set since UDS improves $J(\pi^{\text{eff}}_{\beta})$.

(3) Large unlabeled datasets for long-horizon tasks. Another scenario when UDS relabeling will be beneficial to do compared to not using the unlabeled data at all is in long-horizon tasks (large value of $1/(1-\gamma)$), where a lot of unlabeled data is available, while the labeled dataset, $\mathcal{D}_{\text{U}}$ is relatively very small. Define $H = \frac{1}{1-\gamma}$; then in the case that $|\mathcal{D}^{\text{eff}}(\boldsymbol{s})| = \Omega(H^2)|\mathcal{D}_{\text{L}}(\boldsymbol{s})|$, the sampling error term (term (b)) will consist of one less factor of $1/(1-\gamma)$ when utilizing unlabeled data, compared to when it is not. Since the sampling error grows quadratically in the horizon, whereas the reward bias only grow linearly, a reduction in this term by increasing $|\mathcal{D}^{\text{eff}}(\boldsymbol{s})|$ (i.e., denominator) can improve compared to only using the labeled data, $\mathcal{D}_{\text{L}}$. Thus, even though the rewards may be biased, the addition of large amounts of unlabeled, diverse data in long-horizon tasks can help despite the reward bias incurred.

We empirically verify that UDS indeed helps in the special cases discussed above in Section 7.3.4. However, there are also cases where UDS does not help because of large suboptimality induced due to term (a). In Table 15, we present a summary of the scenarios under which we expect UDS will help or hurt performance. As an example of the latter, when the amount of unlabeled data, $\mathcal{D}_{\text{U}}$ is not very large compared to the labeled dataset, such that a decrease in sampling error does not outweigh the suboptimality induced by reward bias, we should not expect UDS to attain very good performance, which we empirically show in Appendix E.6.2. To handle such cases, our key idea is to re-weight the unlabeled dataset before using it for offline RL training reduce the sampling error and reward bias.

7.3.2.2 *Reducing Reward Bias via CDS*

As discussed above, one way to reduce the suboptimality induced due to reward bias is by preferentially reweighting transitions in the unlabeled data. We would hope that such a scheme can provide a favorable bias-sampling error tradeoff, even though it reduces sample size. In this section, we will derive an optimized reweighting scheme that attains a favorable tradeoff. Intuitively, this optimized scheme suggests that reweighting must reduce distributional shift against the state-action marginal of the policy obtained by running offline RL on only the transitions in the labeled data. This scheme intuitively matches the conservative data sharing (CDS) method from the previous section. We now show that CDS reduces reward bias, controls distributional shift that appears in the numerator of sampling error, and increases the sample size.

Formally, we will derive this reweighting scheme from the perspective of optimizing the effective behavior policy, π^{eff}_{β} induced by the dataset $\mathcal{D}$ after preferentially sharing

transitions from the unlabeled data, so as to minimize reward bias and sampling error, while improving the dataset quality. For understanding purposes, we begin by deriving the choice of π_β^{eff} that reduces only the reward bias component:

Theorem 7.3.2 (Optimized reward bias reduction). *The optimal effective behavior policy that minimizes the bias (a) in Theorem 7.3.1,* $\text{RewardBias}(\pi, \pi_\beta^{\text{eff}})$*, satisfies*

$$\widehat{d}^{\pi_\beta^{\text{eff}}}(\boldsymbol{s}, \boldsymbol{a}) \propto \sqrt{d_{\text{L}}(\boldsymbol{s}, \boldsymbol{a}) d^\pi(\boldsymbol{s}, \boldsymbol{a})},$$

where d^π denotes the state-action marginal of a policy π, and $d_{\text{L}}(\boldsymbol{s}, \boldsymbol{a})$ denotes the density of state-action pair $(\boldsymbol{s}, \boldsymbol{a})$ under the labeled dataset.

A proof of Theorem 7.3.2 is provided in Appendix E.5.2. The expression implies that the state-action marginal of the effective behavior policy (i.e., the rebalanced dataset distribution) must place mass on state-action tuples that are both likely to under the learned policy d^π and appear frequently in the distribution induced by the labeled dataset $\widehat{d}_{\text{L}}$. However, note that Theorem 7.3.2 only accounts for the reward bias and not the other terms pertaining to sampling error and the performance of the effective behavior policy that appear in Theorem 7.3.1. To address both of these issues, in our next theoretical result, Theorem 7.3.3, we derive the reweighting distribution that controls all the terms.

Theorem 7.3.3 (Optimized reweighting unlabeled data; Informal**).** *The optimal effective behavior policy that maximizes a lower bound on $J(\pi_\beta^{\text{eff}}) - [(a) + (b)]$ in Theorem 7.3.1 satisfies $\widehat{d}^{\pi_\beta^{\text{eff}}}(\boldsymbol{s}, \boldsymbol{a}) = p^*(\boldsymbol{s}, \boldsymbol{a})$, where:*

$$p^* = \arg\min_{p \in \Delta^{|\mathcal{S}||\mathcal{A}|}} \sum_{\boldsymbol{s}, \boldsymbol{a}} C_1 \frac{\widehat{d}^\pi(\boldsymbol{s}, \boldsymbol{a})}{\sqrt{p(\boldsymbol{s}, \boldsymbol{a})}} + C_2 |\widehat{d}_{\text{L}}(\boldsymbol{s}, \boldsymbol{a})| \frac{\widehat{d}^\pi(\boldsymbol{s}, \boldsymbol{a})}{p(\boldsymbol{s}, \boldsymbol{a})},$$

where C_1, C_2 are universal positive constants that depend on the sizes of the datasets.

A proof of Theorem 7.3.3 along with a more formal statement listing the constants C_1 and C_2 is provided in in Appendix E.5.3. The first term in the optimization objective of p^* appearing above arises from the sampling error term, while the second term corresponds to the reward bias term in Theorem 7.3.1. The optimal solution for p^* must place high density on $(\boldsymbol{s}, \boldsymbol{a})$ pairs where $\widehat{d}^\pi(\boldsymbol{s}, \boldsymbol{a})$ is high, because this would reduce the objective in the optimization problem above. This corroborates the analysis of Yu et al. [366], which shows that utilizing a reweighting scheme that reduces distributional shift (i.e., makes $\pi(\boldsymbol{a}|\boldsymbol{s}) / \widehat{\pi}_\beta^{\text{eff}}(\boldsymbol{a}|\boldsymbol{s})$ or equivalently, as we find, making $\widehat{d}^\pi(\boldsymbol{s}, \boldsymbol{a}) / \widehat{d}^{\pi_\beta^{\text{eff}}}(\boldsymbol{s}, \boldsymbol{a})$ smaller) will control sampling error, and give rise to performance benefits. In addition, as also shown in Theorem 7.3.2, the reward bias term is also controlled when low distributional shift

Table 16: Results on single-task environments hopper and AntMaze from the D4RL [92] benchmark. The numbers are averaged over three random seeds. We only bold the best-performing method that does not have access to the true reward for relabeling. UDS outperforms No Sharing in three out of the four settings while achieving competitive performances compared to Sharing All in all the settings. CDS+UDS further improves over UDS in three out of four settings.

Environment	Labeled data	Unlabeled data	CDS+UDS	UDS	No Sharing	Reward Pred.	VICE	RCE	Oracle Methods CDS	Oracle Methods Sharing All
D4RL hopper	expert	random	**81.5**	78.6	77.1	67.6	n/a	n/a	83.3	86.1
	expert	medium	**78.3**	64.4	77.1	51.7	n/a	n/a	82.5	64.6
D4RL AntMaze	expert	medium-play	82.6	**82.7**	17.2	0.0	0.0	0.0	83.5	83.1
	expert	large-play	**47.1**	33.1	0.7	0.0	0.0	0.0	46.1	50.2

appears. This means that rebalancing the dataset to control distributional shift between the learned policy $\widehat{d}^{\pi}(\boldsymbol{s}, \boldsymbol{a})$ and $\widehat{d}^{\pi^{\text{eff}}_{\beta}}(\boldsymbol{s}, \boldsymbol{a})$ is effective in unlabeled settings as well.

While the scheme derived above can, in principle, be implemented exactly in practice, it would require computing state-marginals. Since, computing state marginals accurately in an offline setting has been an outstanding challenge, we instead can utilize a reweighting scheme that corrects for distributional shift approximately without needing state-marginals. One of such methods is conservative data sharing (CDS) [366] that can be implemented without requiring additional components beyond the machinery of the offline RL method. Formally, CDS is given by:

$$\mathcal{D}_i^{\text{eff}} = \mathcal{D}_i \cup (\cup_{j \neq i} \{(\boldsymbol{s}_j, \boldsymbol{a}_j, \boldsymbol{s}'_j, r_i) \in \mathcal{D}_{j \to i} : \Delta^{\pi}(\boldsymbol{s}, \boldsymbol{a}) \geq 0\}),$$

where $\boldsymbol{s}_j, \boldsymbol{a}_j, \boldsymbol{s}'_j$ denote the transition from $\mathcal{D}_j$, r_i denotes the reward of $\boldsymbol{s}_j, \boldsymbol{a}_j, \boldsymbol{s}'_j$ relabeled for task i, π denotes the task-conditioned policy $\pi(\cdot|\cdot, i)$, $\Delta^{\pi}(\boldsymbol{s}_j, \boldsymbol{a}_j)$ is the condition that shares data only if the expected conservative Q-value of the relabeled transition exceeds the top k-percentile of the conservative Q-values of the original data. In our experiments and analysis in Section 7.3.3, we find that utilizing CDS improves performance over simply training with UDS in a number of domains supporting the theoretical analysis.

7.3.2.3 *Why Can We Expect UDS to Outperform Reward Prediction Methods?*

While we have discussed various conditions where we would expect UDS (with or without various reweighting methods) to improve final performance over an approach that does not utilize the unlabeled data, it is also instructive to consider how it comes to a method that instead trains an approximate reward function, $\widehat{r}_{\phi}(\boldsymbol{s}, \boldsymbol{a})$ using the labeled data, and then uses this approximate reward to annotate the unlabeled data. In our experiments, we will show, perhaps surprisingly that UDS often outperforms prior reward learning methods. In this section, we provide some intuition for why. First, we note the following generic expression for reward bias (term (a) in Theorem 7.3.1) for any approach:

$$\text{RewardBias}(\pi, \pi^{\text{eff}}_{\beta}) = \frac{\sum_{\boldsymbol{s},\boldsymbol{a}} \Delta\left(\widehat{d}^{\pi_{\beta}{}^{\text{eff}}}, \widehat{d}^{\pi}\right) \cdot \Delta r(\boldsymbol{s}, \boldsymbol{a})}{1 - \gamma},$$

where $\Delta r(\boldsymbol{s}, \boldsymbol{a})$ is the error in the reward applied to the unlabeled data, such that $\Delta r(\boldsymbol{s}, \boldsymbol{a}) = (1 - f(\boldsymbol{s}, \boldsymbol{a}))r(\boldsymbol{s}, \boldsymbol{a})$ for UDS, and $\Delta r(\boldsymbol{s}, \boldsymbol{a}) = r(\boldsymbol{s}, \boldsymbol{a}) - \widehat{r}_\phi(\boldsymbol{s}, \boldsymbol{a})$ for a reward prediction method. Note that while for UDS, $\forall\ \boldsymbol{s}, \boldsymbol{a},\ \Delta r(\boldsymbol{s}, \boldsymbol{a}) \geq 0$, this is not necessarily the case for a generic reward prediction method.

UDS. Since $\Delta r(\boldsymbol{s}, \boldsymbol{a}) \geq 0$ for all state-action tuples, state-action pairs that appear more frequently under the learned policy compared to the effective behavior policy, i.e., when $\Delta\left(\widehat{d}^{\pi_\beta{}^{\text{eff}}}, \widehat{d}^{\pi}\right) < 0$, contribute to reducing the suboptimality from reward bias.

Reward prediction. When $\Delta r(\boldsymbol{s}, \boldsymbol{a}) = r(\boldsymbol{s}, \boldsymbol{a}) - \widehat{r}_\phi(\boldsymbol{s}, \boldsymbol{a})$, $\Delta r(\boldsymbol{s}, \boldsymbol{a})$ may not be positive on all state-action tuples, and thus reward prediction methods fail to reduce the contribution of such state-action pairs in term (a). Since policy optimization will seek out those policies that maximize $\widehat{d}^{\pi}(\boldsymbol{s}, \boldsymbol{a})$ on state-action pairs with high rewards, state-action pairs where $\Delta(\widehat{d}^{\pi_\beta^{\text{eff}}}, \widehat{d}^{\pi}) < 0$ (i.e., state-action pairs that appear more under the learned policy) are likely to also have $\Delta r(\boldsymbol{s}, \boldsymbol{a}) < 0$. This "exploitation" inhibits the correction of reward bias and provides an explanation for why reward prediction approaches may still not perform well due to overestimation errors in the reward function.

7.3.3 *Experimental Evaluation of UDS and CDS+UDS*

In our experiments, we aim to evaluate whether the theoretical potential for simple minimum-reward relabeling to attain good results is reflected in benchmark tasks and more complex offline RL settings. In particular, we will study: **(1)** can UDS match or exceed the performance of sophisticated reward inference methods and methods with oracle reward functions in simulated locomotion and navigation tasks? **(2)** can combining with CDS further improve the results achieved by UDS?, **(3)** how does UDS behave with and without combining with an optimized reweighting strategy in multi-task offline RL settings?, **(4)** under which conditions does UDS work and not work and does optimizing for reweighting via CDS help?

Comparisons. We evaluate multiple approaches alongside with **UDS** and **CDS+UDS**: **No Sharing**, which uses the labeled data only without using any of the unlabeled data, **Reward Predictor**, which is trained via supervised classification or regression to directly predict the reward in sparse and dense reward settings respectively, **VICE** [89] and **RCE** [77], inverse RL methods only applicable to sparse reward settings, that learn either a reward or Q-function classifier from both the labeled data and unlabeled data (treated as negatives) and then annotate the unlabeled data with the learned classifier, In the multi-task setting, we modify **VICE** and **RCE** accordingly by extracting transitions with reward labels equal to 1 and treating these datapoints as positives to learn the classifier for each task. We also train **No Sharing**, **Reward Predictor**, **VICE** and **RCE**, but adapt them to the offline setting using CQL, i.e. the same underlying offline RL method as in **UDS**

and **CDS+UDS**. For more details on experimental set-up and hyperparameter settings, please see Appendix E.11. We also include evaluations of our methods under different quality of the relabeled data in Appendix E.7 and comparisons to model-based offline RL approaches in Appendix E.9 and prior methods [354, 355] that leverage unlabeled data for representation learning instead of sharing directly in Appendix E.10.

Datasets and tasks. To answer questions (1) and (2), we perform empirical evaluations on two state-based single-task locomotion datasets. To answer question (3), we further evaluate all methods on three state-based multi-task datasets that consist of robotic manipulation, navigation and locomotion domains respectively and one image-based multi-task manipulation dataset from Section 7.2.5.

Single-task domains & datasets. We adopt two environments: hopper and the AntMaze from the D4RL benchmark [92] for evaluation in the single-task setting. For the hopper environment, we consider two scenarios where we have 10k labeled data from the hopper-expert and 1M unlabeled transitions from the hopper-random datasets and hopper-medium datasets respectively. This setup is resembles practical problems where unlabeled data is of low-quality and even, irrelevant to the target task. For AntMaze, following the setup in [355], we use 10k expert transitions as the labeled data and the entire datasets of large-play and medium-play as the unlabeled data.

Table 17: We perform an empirical analysis on the single-task environment hopper from D4RL [92] benchmark to test the sensitivity of UDS under the data quality and data coverage for both the labeled task data and unlabeled data. The numbers are averaged over ten random seeds. We bold the best method without true reward relabeling. UDS outperforms No Sharing in 5 out of 6 settings while achieving competitive performances compared to Sharing All in 5 out of 6 settings. CDS+UDS is able to further improve over UDS significantly in such a setting and also outperforms UDS in 5 out of 6 settings in general.

Environment	Labeled data type / size	Unlabeled data type / size	CDS+UDS	UDS	No Sharing	Oracle Methods	
						CDS	Sharing All
D4RL hopper	**(a)** expert / 10k transitions	random / 1M transitions	**82.1**	78.8	77.1	83.3	86.1
	(b) expert / 10k transitions	medium / 1M transitions	**78.1**	64.8	77.1	82.5	64.6
	(c) expert / 10k transitions	expert / 990k transitions	106.1	**108.4**	77.1	106.6	112.3
	(d) medium / 10k transitions	random / 1M transitions	**33.2**	9.9	28.7	41.2	38.9
	(e) medium / 10k transitions	expert / 1M transitions	**108.9**	106.7	28.7	111.1	107.5
	(f) random / 10k transitions	medium / 1M transitions	**63.5**	47.1	9.6	92.3	69.8
	(g) random / 10k transitions	expert / 1M transitions	**101.2**	95.9	9.6	110.9	102.8

Multi-task domains & datasets. We consider several multi-task domains. The first set of domains are from Section 7.2.5. We also evaluate on the multi-task walker environment with dense rewards, which we will discuss in detail in Appendix E.8. In addition, we test multi-task offline RL methods with UDS in the multi-task visual manipulation domain, which provides a realistic scenario, of the sorts we will encounter in robotic settings in practice. In this domain, there are 10 tasks with different combinations of object-oriented grasping, with 7 objects (banana, bottle, sausage, milk box, food box, can and carrot), as well as placing the picked objects onto one of three fixtures (bowl, plate and divider

plate). For domains except the walker environment, we use binary rewards, where 1 denotes the successful completion of the task and 0 corresponds to failure. For details, see Appendix E.11.2.

Table 18: Results for multi-task robotic manipulation (Meta-World) and navigation environments (AntMaze) with low-dimensional state inputs. Numbers are averaged across 6 seeds, ± the 95%-confidence interval. We take the results of No Sharing, Sharing All and CDS, directly from [366]. We bold the best-performing method that does not have access to the true rewards for relabeling. We include per-task performance for Meta-World domains and the overall performance averaged across tasks (highlighted in gray) for all three domains. Both CDS+UDS and UDS outperforms prior vanilla multi-task offline RL approach (No Sharing) and reward learning methods (Reward Predictor, VICE and RCE) while performing competitively compared to oracle reward relabeling methods.

Environment	Tasks	CDS+UDS	UDS	VICE	RCE	No Sharing	Reward Pred.	Oracle Methods CDS	Oracle Methods Sharing All
Meta-World	door open	**61.3%**±7.9%	51.9%±25.3%	0.0%±0.0%	0.0%±0.0%	14.5%±12.7%	0.0%±0.0%	58.4%±9.3%	34.3%±17.9%
	door close	54.0% ±42.5%	12.3%±27.6%	66.7%%±47.1%	0.0%±0.0%	4.0%±6.1%	**99.3%**±0.9%	65.3%±27.7%	48.3%±27.3%
	drawer open	**73.5%**±9.6%	61.8%±16.3%	0.0%±0.0%	0.0%±0.0%	16.0%±17.5%	13.3%±18.9%	57.9%±16.2%	55.1%±9.4%
	drawer close	99.3%±0.7%	**99.6%**±0.7%	19.3%±27.3%	2.7%±1.7%	99.0%±0.7%	50.3%±35.8%	99.0%±0.7%	98.8%±0.7%
	average	**71.2%** ± 11.3%	56.4%±12.8%	21.5%±0.7%	0.7%±0.4%	33.4%±8.3%	41.0%±11.9%	70.1%±8.1%	59.4%±5.7%
AntMaze	medium (3 tasks)	**31.5%**±3.0%	26.5%±9.1%	2.9%±1.0%	0.0%±0.0%	21.6%±7.1%	3.8%±3.8%	36.7%±6.2%	22.9%±3.6%
	large (7 tasks)	**18.4%**±6.1%	14.2%±3.9%	2.5%±1.1%	0.0%±0.0%	13.3% ± 8.6%	5.9%±4.1%	22.8% ± 4.5%	16.7% ± 7.0%

Table 19: Results for multi-task imaged-based robotic manipulation domains in [366]. Numbers are averaged across 3 seeds, ± the 95% confidence interval. UDS outperforms No Sharing in 7 out of 10 tasks as well as the average task performance, while performing comparably to Sharing All. CDS+UDS further improves the performance of UDS and outperforms No Sharing in all of the 10 tasks.

Task Name	CDS+UDS	UDS	No Sharing	CDS (oracle)	Sharing All (oracle)
lift-banana	**55.9%**±11.7%	48.6%±5.1%	20.0%±6.0%	**53.1%**±3.2%	41.8%±4.2%
lift-bottle	**72.9%**±12.8%	58.1%±3.6%	49.7%±8.7%	**74.0%**±6.3%	60.1%±10.2%
lift-sausage	**74.3%**±8.3%	66.8% ± 2.7%	60.9%±6.6%	**71.8%**±3.9%	70.0%±7.0%
lift-milk	73.5%±6.7%	**74.5%**±2.5%	68.4%±6.1%	**83.4%**±5.2%	72.5%±5.3%
lift-food	**66.3%**±8.3%	53.8%±8.8%	39.1%±7.0%	**61.4%**±9.5%	58.5%±7.0%
lift-can	**64.9%**±7.1%	61.0%±6.8%	49.1%±9.8%	**65.5%**±6.9%	57.7%±7.2%
lift-carrot	**84.1%**±3.6%	73.4%±5.8%	69.4%±7.6%	**83.8%**±3.5%	75.2%±7.6%
place-bowl	**83.4%**±3.6%	77.6%±1.6%	80.3%±8.6%	**81.0%**±8.1%	70.8%±7.8%
place-plate	**86.2%**±1.8%	78.7%±2.2%	86.1%±7.7%	**85.8%**±6.6%	78.7%±7.6%
place-divider-plate	**89.0%**±2.2%	80.2%±2.2%	85.0%±5.9%	**87.8%**±7.6%	79.2%±6.3%
average	**75.0%**±3.3%	67.3%±0.8%	60.8%±7.5%	**74.8%** ±6.4%	66.4%±7.2%

7.3.3.1 *Results of Empirical Evaluations*

Results of Question (1) and (2). We evaluate each method on the single-task hopper and AntMaze domains. As shown in Table 16, we find that UDS outperforms No Sharing in 3 out of the 4 settings and reward learning methods in all the settings. We hypothesize that reward learning methods underperform because reward predictors are unable to achieve reasonable generalization ability in the limited labeled data setting. Note that UDS even achieves competitive performance as the oracle Sharing All method. Furthermore, an optimized reweighting scheme, i.e., CDS+UDS, is able to improve over UDS in each case,

including cases where UDS fails to improve over No Sharing, indicating a large reward bias. These results testify to the effectiveness of optimizing for reweighting when dealing with unlabeled data. Note that on the AntMaze domain, CDS+UDS performs comparably to the recent approach [355] that performs representation learning on the offline dataset first and then run offline training leveraging the learned representation. CDS+UDS also outperforms all the prior methods with learned representations discussed in [355].

Results of Question (3). Observe in Table 18 that UDS outperforms naïve multi-task offline RL without data sharing and reward learning methods, suggesting that leveraging unlabeled data with our simple method UDS can boost offline RL performance in both multi-task manipulation and navigation domains. Since reward learning approaches obtain similar or worse results compared to No Sharing, which we suspect could be due to erroneous predictions on unseen transitions in the multi-task data, we only compare our methods to No Sharing and the oracle methods in the image-based experiments. As shown in Table 19, UDS outperforms No Sharing in 7 out of 10 tasks as well as the average task performance by a significant margin. Therefore, UDS is able to effectively leverage unlabeled data from other tasks and achieves potentially surprisingly strong results compared to more sophisticated methods that handle unlabeled offline data. We also find that CDS+UDS further improves upon the performance of UDS, which indicates that optimizing for reweighting via CDS+UDS can actually work well.

Finally, note that, CDS+UDS and UDS attain performance competitive with their oracle counterparts, CDS and Sharing All, that assume access to full reward information, both in Tables 18 & 19. This is potentially very surprising, and one could hypothesize that this might be because most transitions in the unlabeled dataset were actually failures, and hence, were labeled correctly by UDS. However, to the contrary, our diagnostic analysis in Table 55, Appendix E.7 reveals that the unlabeled data *does not* consist entirely of failures; in fact, around 60% of the unlabeled data succeeds at the task of interest, indicating that the rewards are wrong for more than half the unlabeled data. In spite of this, UDS and CDS+UDS perform well. This indicates that UDS can be effective in removing the dependence of functional form of reward functions, which is often costly, without much loss in the performance and CDS+UDS can boost performance by reducing the bias.

7.3.4 *Empirical Analysis of UDS and CDS+UDS*

To answer question (4), we analyze UDS and CDS+UDS on several offline RL problems designed to test robustness and sensitivity to the data coverage and the data quality on the single-task hopper domain. To create these instances, we consider 7 different combinations of data quality (hopper-expert, hopper-medium or hopper-random) and amount of labeled and unlabeled data labeled as **(a)-(g)** in Table 17. In Appendix E.6.1, we present results for a similar analysis in the multi-task Meta-World setting. We evaluate

UDS, CDS+UDS and No Sharing in each case, and also present results for CDS and Sharing All approaches which assume access to the actual reward, for understanding. Additionally, we conduct an ablation that compares UDS and CDS+UDS to reward learning methods in settings where the labeled data size and quality are varied, which is included in Appendix E.6.3.

When the labeled data is of expert-level, adding unlabeled random or medium data (cases **a** and **b** in Table 17) should only increase coverage, since the labeled data only consists of expert transitions. Moreover, the reward bias induced due to incorrect labeling of rewards on the medium unlabeled data should not hurt, since the 10k expert transitions retain their correct labels, and the medium/random data should only serve as negatives, provided the annotated rewards on this data are worse than the rewards in the expert data. Therefore, we expect the benefits of coverage to outweigh any reward bias; as seen in Table 17, we find that UDS indeed helps compared to No Sharing. In particular, in those two cases, UDS approaches the oracle Sharing All method.

Since reward bias is exacerbated when the unlabeled data is of higher quality and present in large amounts (cases **e, f, g**), it is reasonable to surmise that UDS will perform worse in such scenarios. However, on the contrary, in settings: random labeled data with medium or expert unlabeled data and medium labeled data with expert unlabeled data, we find that even though the rewards on the unlabeled transitions are incorrect, the addition of unlabeled data into training improves performance. This is because a higher quality unlabeled data improves the performance of the effective behavior policy, thereby improving performance for a conservative offline RL method. This result validates Theorem 7.3.1. We also conduct an ablation that varies the amount of unlabeled data in Appendix E.6.2. This ablation shows that the benefit of UDS reduces as the we reduce the amount of unlabeled data, which confirms our theory.

In the case where labeled data is medium and unlabeled data is random (case **d**), we find that UDS hurts compared to No Sharing. This is because the labeled data as well as unlabeled data are both low-medium quality and medium data already provides decent coverage (not as high as random data, but not as low as expert data). Therefore, in this case, we believe that the addition of unlabeled data neither provides trajectories of good quality that can help improve performance, nor does it significantly improve coverage, and only hurts by incurring reward bias. We therefore believe that UDS may not help in such cases where the coverage does not improve, and added data is not so high quality, which also agrees with our theoretical analysis. Furthermore, in the case where the labeled and unlabeled data are both expert (case **c**), UDS performs close to oracle methods CDS and Sharing All, which confirms the insights derived from Section 7.3.2.1 that UDS does not introduce bias when the labeled and unlabeled data have the same distribution and hence should perform well.

Finally, note that CDS+UDS is able to improve over UDS in 5 out of 6 settings in hopper, suggesting that CDS+UDS gains benefit from reducing the reward bias as well as the sampling error shown in Section 7.3.2.

7.4 DISCUSSION AND LIMITATIONS

In this chapter, we studied the problem of *data sharing* in multi-task offline RL, when presented with diverse, unlabeled data. We developed approaches to tackle two facets of this problem: (i) learning to selectively reweight multi-task data to guarantee a performance improvement, and (ii) learning in the presence of no reward annotations. Our approaches CDS and UDS are theoretically-motivated and lead to significant boost in performance in several multi-task problem scenarios, where we must learn from diverse datasets. We provide a theoretical and empirical analysis in simplified setups to understand the conditions under which our approaches can work and find that, in general, when reward annotations are not present, combining CDS and UDS together can be performant. We also showed that UDS can also be utilized for effectively incorporating unlabeled data in single-task offline RL settings. Overall, we believe that this chapter justifies the effectiveness of simple methods for using unlabeled and multi-task data in offline RL and shed light on directions for future work that can better control the reward bias and enjoy better policy performance.

While CDS and UDS provide an initial way to incorporate diverse, multi-task datasets in offline RL, these approaches do not tackle several other challenges that often come with real datasets. For instance, a running assumption in this chapter is that the state and action spaces for transitions in the task-specific data and the diverse data are identical, but this assumption is not specified in many practical problems, an example of which is incorporating diverse human videos for downstream robotic control. From a theoretical standpoint, takeaways from our analyis can be improved to show policy optimality guarantees. We leave these as avenues for future work.

ACKNOWLEDGEMENTS AND FUNDING

We thank Kanishka Rao, Julian Ibarz, Xinyang Geng, Avi Singh, members of IRIS at Stanford, RAIL at UC Berkeley and Robotics at Google and Google Research for valuable and constructive feedback on an early version of this manuscript. This research was funded in part by Google, ONR grants N00014-21-1-2685, N00014-20-1-2675, and N00014-21-1-2838, Intel Corporation and the DARPA Assured Autonomy Program. CF is a CIFAR Fellow in the Learning in Machines and Brains program.

8

ONLINE RL FINE-TUNING OF OFFLINE RL

Abstract

A compelling use case of offline RL is to obtain a policy initialization from existing datasets followed by fast online fine-tuning with limited interaction. However, a number of offline RL methods tend to behave poorly during fine-tuning. In this chapter, we study the fine-tuning problem in the context of conservative value estimation methods and we devise an approach for learning an effective initialization from offline data that also enables fast online fine-tuning capabilities. Our approach, calibrated Q-learning (Cal-QL), accomplishes this by learning a conservative value function initialization that underestimates the value of the learned policy from offline data, while also ensuring that the learned Q-values are at a reasonable scale. We refer to this property as *calibration*, and define it mathematically as providing a lower bound on the true value function of the learned policy and an upper bound on the value of some other (suboptimal) reference policy, which may simply be the behavior policy. We show that a conservative offline RL algorithm that also learns a calibrated value function leads to effective online fine-tuning, enabling us to take the benefits of offline initializations in online fine-tuning.

8.1 INTRODUCTION

Modern machine learning successes follow a common recipe: pre-training models on general-purpose, Internet-scale data, followed by fine-tuning the pre-trained initialization on a limited amount of data for the task of interest [132, 62]. How can we translate such a recipe to sequential decision-making problems? A natural way to instantiate this paradigm is to utilize offline RL for initializing value functions and policies from static datasets, followed by online fine-tuning to improve this initialization with limited active interaction. If successful, such a recipe might enable effective online RL with much fewer samples than current methods that learn from scratch.

Many algorithms for offline RL have been applied to online fine-tuning. Empirical results across such works suggest a counter-intuitive trend: policy initializations obtained from more effective offline RL methods tend to exhibit worse online fine-tuning performance, even within the same task (see Table 2 of [176] & Figure 4 of [345]). On the other end, online RL methods training from scratch (or RL from demonstrations [331], where the replay buffer is seeded with the offline data) seem to improve online at a significantly faster rate. But these online methods require actively collecting data by rolling out policies from scratch, which inherits similar limitations to naïve online RL methods in problems where data collection is expensive or dangerous. Overall, these results suggest that it is challenging to devise an offline RL algorithm that both acquires a good initialization from prior data and also enables efficient fine-tuning.

How can we devise a method to learn an effective policy initialization that also improves during fine-tuning? We have shown that one can learn a good offline initialization by optimizing the policy against a *conservative* value function obtained from an offline dataset. But, as we show in Section 8.4.1, conservatism alone is insufficient for efficient online fine-tuning. Conservative methods often tend to "unlearn" the policy initialization learned from offline data and waste samples collected via online interaction in recovering this initialization. We find that the "unlearning" phenomenon is a consequence of the fact that value estimates produced via conservative methods can be significantly lower than the ground-truth return of *any* valid policy. Having Q-value estimates that do not lie on a similar scale as the return of a valid policy is problematic. Because once fine-tuning begins, actions executed in the environment for exploration that are actually worse than the policy learned from offline data could erroneously appear better, if their ground-truth return value is larger than the learned conservative value estimate. Hence, subsequent policy optimization will degrade the policy performance until the method recovers.

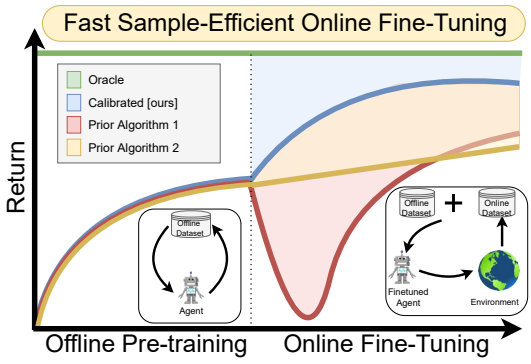


Figure 30: We study **offline RL pre-training followed by online RL fine-tuning**. Some prior offline RL methods tend to exhibit slow performance improvement in this setting (yellow), resulting in worse asymptotic performance. Others suffer from initial performance degradation once online fine-tuning begins (red), resulting in a high cumulative regret. We develop an approach that "*calibrates*" the value function to attain a fast improvement with a smaller regret (blue).

If we can ensure that the conservative value estimates learned using the offline data are *calibrated*, meaning that these estimates are on a similar scale as the true return values, then we can avoid the unlearning phenomenon caused by conservative methods (see the formal definition in 8.4.1). Of course, we cannot enforce such a condition perfectly, since

it would require eliminating all errors in the value function. Instead, we devise a method for ensuring that the learned values upper bound the true values of some *reference policy* whose values can be estimated more easily (e.g., the behavior policy), while still lower bounding the values of the learned policy. Though this does not perfectly ensure that the learned values are correct, we show that it still leads to sample-efficient online fine-tuning. Thus, our practical method, **calibrated Q-learning (Cal-QL)**, learns conservative value functions that are "calibrated" against the behavior policy, via a simple modification to existing conservative methods.

The main contribution of this chapter is Cal-QL, a method for acquiring an offline initialization that facilitates online fine-tuning. Cal-QL aims to learn conservative value functions that are calibrated with respect to a reference policy (e.g., the behavior policy). Our analysis of Cal-QL shows that Cal-QL attains stronger guarantees on cumulative regret during fine-tuning. In practice, Cal-QL can be implemented on top of conservative Q-learning [181], a prior offline RL method, without any additional hyperparameters. We evaluate Cal-QL across a range of benchmark tasks from [92], [303] and [240], including robotic manipulation and navigation. We show that Cal-QL matches or outperforms the best methods on all tasks, in some cases by 30-40%.

8.2 Related Work

Several prior works suggest that online RL methods typically require a large number of samples [297, 333, 362, 157, 371, 120, 208] to learn from scratch. We can utilize offline data to accelerate online RL algorithms. Prior works do this in a variety of ways: incorporating the offline data into the replay buffer of online RL [283, 331, 138, 308], utilizing auxiliary behavioral cloning losses with policy gradients [273, 163, 376, 375], or extracting a high-level skill space for downstream online RL [119, 9]. While these methods improve the sample efficiency of online RL from scratch, as we will also show in our results, they do not eliminate the need to actively roll out poor policies for data collection.

To address this issue, a different line of work first runs offline RL for learning a good policy and value initialization from the offline data, followed by online fine-tuning [241, 175, 221, 21, 342, 200, 228]. These approaches typically employ offline RL methods based on policy constraints or pessimism [95, 296, 118, 105, 175, 303, 200] on the offline data, then continue training with the same method on a combination of offline and online data once fine-tuning begins [238, 166, 365, 181, 35]. Although pessimism is crucial for offline RL [152, 47], using pessimism or constraints for fine-tuning [241, 175, 221] slows down fine-tuning or leads to initial unlearning, as we will show in Section 8.4.1. In effect, these prior methods either fail to improve as fast as online RL or lose the initialization from offline RL. We aim to address this limitation by understanding some conditions on the offline initialization that enable fast fine-tuning.

Our approach is most related to methods that utilize a pessimistic offline RL algorithm for offline training but incorporate exploration in fine-tuning [200, 228, 342]. In contrast to these works, our method aims to learn a better offline initialization that enables standard online fine-tuning. Our approach fine-tunes naïvely without ensembles [200] or exploration [228] and, as we show in our experiments, this alone is enough to outperform approaches that employ explicit optimism during data collection.

8.3 Problem Statement and Additional Notation

Given access to an offline dataset $\mathcal{D} = \{(s, a, r, s')\}$ collected using a behavior policy π_β, we aim to first train a good policy and value function using the offline dataset $\mathcal{D}$ alone, followed by an online phase that utilizes online interaction in $\mathcal{M}$. Our goal during fine-tuning is to obtain the optimal policy with the smallest number of online samples. This can be expressed as minimizing the **cumulative regret** over rounds of online interaction:

$$\text{Reg}(K) := \mathbb{E}_{s\sim\rho} \sum_{k=1}^{K} \left[V^\star(s) - V^{\pi^k}(s) \right]. \tag{8.3.1}$$

8.4 When can offline RL initializations enable fast online fine-tuning?

A starting point for offline pre-training and online fine-tuning is to simply initialize the value function with one that is produced by an existing offline RL method and then perform fine-tuning. However, we empirically find that initializations learned by many offline RL algorithms can perform poorly during fine-tuning. We will study the reasons for this poor performance for the subset of conservative methods to motivate and develop our approach for online fine-tuning, calibrated Q-learning.

8.4.1 *Empirical Analysis*

Offline RL followed by online fine-tuning typically poses non-trivial challenges for a variety of methods. While analysis in prior work [241] notes challenges for a subset of offline RL methods, in Figure 31, we evaluate the fine-tuning performance of a variety of prior offline RL methods (CQL [181], IQL [175], TD3+BC [94], AWAC [241]) on a particular diagnostic instance of a visual pick-and-place task with a distractor object and sparse binary rewards [303], and find that all methods struggle to attain the best possible performance, quickly. More details about this task are in Appendix F.4.

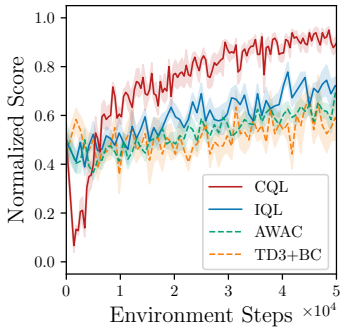


Figure 31: Multiple prior offline RL algorithms suffer from difficulties during fine-tuning including poor asymptotic performance and initial unlearning.

While the offline Q-function initialization obtained from all methods attains a similar (normalized) return of around 0.5, they suffer from difficulties during fine-tuning: TD3+BC, IQL, AWAC attain slow asymptotic performance and CQL unlearns the offline initialization, followed by spending a large amount of online interaction to recover the offline performance again, before any further improvement. This initial unlearning appears in multiple tasks as we show in Appendix F.8. In this work, we focus on developing effective fine-tuning strategies on top of conservative value estimation methods like CQL. To do so, we next study the potential reason behind the initial unlearning in CQL.

Why does CQL unlearn initially? To understand why CQL unlearns initially, we inspect the learned Q-values averaged over the dataset in Figure 32. Observe that the Q-values learned by CQL in the offline phase are *much* smaller than their ground-truth value (as expected), but these Q-values drastically jump and adjust in scale when fine-tuning begins. In fact, we observe that performance recovery (red segment in Figure 32) *coincides* with a period where the range of Q-values changes to match the true range. This is as expected: as a conservative Q-function experiences new online data, actions much worse than the offline policy on the rollout states appear to attain higher rewards compared to the highly underestimated offline Q-function, which in turn deceives the policy optimizer into unlearning the initial policy. We illustrate this idea visually in Figure 33. Once the Q-function has adjusted and the range of Q-values closely matches the true range, then fine-tuning can proceed normally, after the dip.

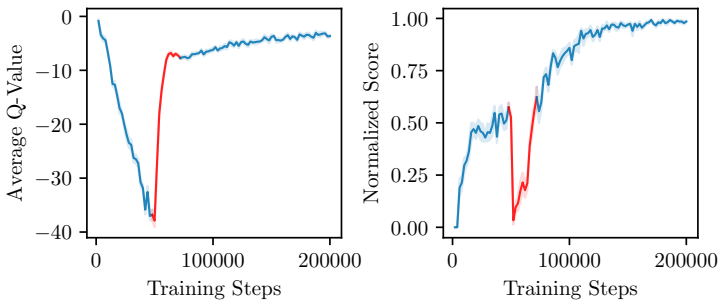


Figure 32: The evolution of the average Q-value and the success rate of CQL over the course of offline pre-training and online fine-tuning. Fine-tuning begins at 50K steps. The red-colored part denotes the period of performance recovery which also coincides with the period of Q-value adjustment.

To summarize, our empirical analysis indicates that methods existing fine-tuning methods suffer from difficulties such as initial unlearning or poor asymptotic performance. In particular, we observed that conservative methods can attain good asymptotic performance, but "waste" samples to correct the learned Q-function. Thus, we attempt to develop a fine-tuning method that builds on top of an existing conservative method, CQL.

8.4.2 *Conditions on the Offline Initialization that Enable Fast Fine-Tuning*

Our observations from the preceding discussion motivate two conclusions in regard to the offline Q-initialization for fast fine-tuning: **(a)** methods that learn **conservative** Q-functions can attain good asymptotic performance, and **(b)** if the learned Q-values

closely match the range of ground-truth Q-values on the task, then online fine-tuning does not need to devote samples to unlearn and then recover the offline initialization. One approach to formalize this intuition of Q-values lying on a similar scale as the ground-truth Q-function is via the requirement that the conservative Q-values learned by the conservative method must be lower-bounded by the Q-value of a sub-optimal reference policy. This will prevent conservatism from learning overly small Q-values. We will refer to this property as "calibration" with respect to the reference policy.

Definition 8.4.1 (Calibration). *An estimated Q-function Q^π_θ for a given policy π is said to be calibrated with respect to a reference policy μ if:*

$$\mathbb{E}_{\boldsymbol{a}\sim\pi}\left[Q^\pi_\theta(\boldsymbol{s},\boldsymbol{a})\right] \geq \mathbb{E}_{\boldsymbol{a}\sim\mu}\left[Q^\mu(\boldsymbol{s},\boldsymbol{a})\right] := V^\mu(s), \forall s \in D. \tag{8.4.1}$$

If the learned Q-function Q^π_θ is calibrated with respect to a policy μ that is worse than π, it would prevent unlearning during fine-tuning that we observed in the case of CQL. This is because the policy optimizer would not unlearn π in favor of a policy that is worse than the reference policy μ upon observing new online data as the expected value of π is constrained to be larger than V^μ: $\mathbb{E}_{\boldsymbol{a}\sim\pi}\left[Q^\pi_\theta(\boldsymbol{s},\boldsymbol{a})\right] \geq V^\mu(\boldsymbol{s})$. Our practical approach Cal-QL will enforce calibration with respect to a policy μ whose ground-truth value, $V^\mu(\boldsymbol{s})$, can be estimated reliably without bootstrapping error (e.g., the behavior policy induced by the dataset). This is the key idea behind our method (as we will discuss next) and is visually illustrated in Figure 33.

8.5 CAL-QL: CALIBRATED Q-LEARNING

Our approach, calibrated Q-learning (Cal-QL) aims to learn a conservative and calibrated value function initializations from an offline dataset. To this end, Cal-QL builds on CQL from Chapter 5 and then constrains the learned Q-function to produce Q-values larger than the Q-value of a reference policy μ per Definition 8.4.1. In principle, our approach can utilize many different choices of reference policies, but for developing a practical method, we simply utilize the behavior policy as our reference policy.

Calibrating CQL. We can constrain the learned Q-function Q^π_θ to be larger than V^μ via a simple change to the CQL training objective from Chapter 5: masking out the push down of the learned Q-value on out-of-distribution (OOD) actions in CQL if the Q-function is not calibrated, i.e., if $\mathbb{E}_{a\sim\pi}\left[Q^\pi_\theta(\boldsymbol{s},\boldsymbol{a})\right] \leq V^\mu(\boldsymbol{s})$. Cal-QL modifies the CQL regularizer, $\mathcal{R}(\theta)$ in this manner:

$$\mathbb{E}_{\boldsymbol{s}\sim\mathcal{D},\boldsymbol{a}\sim\pi}\left[\max\left(Q_\theta(s,a), V^\mu(s)\right)\right] - \mathbb{E}_{\boldsymbol{s},\boldsymbol{a}\sim\mathcal{D}}\left[Q_\theta(\boldsymbol{s},\boldsymbol{a})\right], \tag{8.5.1}$$

where the changes from standard CQL are depicted in red. As long as α in CQL is large, for any state-action pair where the learned Q-value is smaller than Q^μ, the Q-function in Equation 8.5.1 will upper bound Q^μ in a tabular setting. Of course, as with any

practical RL method, with function approximators and gradient-based optimizers, we cannot guarantee that we can enforce this condition for every state-action pair, but in our experiments, we find that Equation 8.5.1 is sufficient to enforce the calibration in expectation over the states in the dataset.

Pseudo-code and implementation details. Our implementation of Cal-QL directly builds on the implementation of CQL from Geng [104]. We present a pseudo-code for Cal-QL in Algorithm 7. Additionally, we list the hyperparameters α for the CQL algorithm and our baselines for each suite of tasks in Appendix F.5. Following the protocol in prior work [175, 308], the practical implementation of Cal-QL trains on a mixture of the offline data and the new online data, weighted in some proportion during fine-tuning. To get $V^{\mu}(s)$, we can fit a function approximator Q^{μ}_{θ} or V^{μ}_{θ} to the return-to-go values via regression, but we observed that also simply utilizing the return-to-go estimates for tasks that end in a terminal was sufficient for our use case. We show in Section 8.7, how this simple change to the objective drastically improves over prior fine-tuning results.

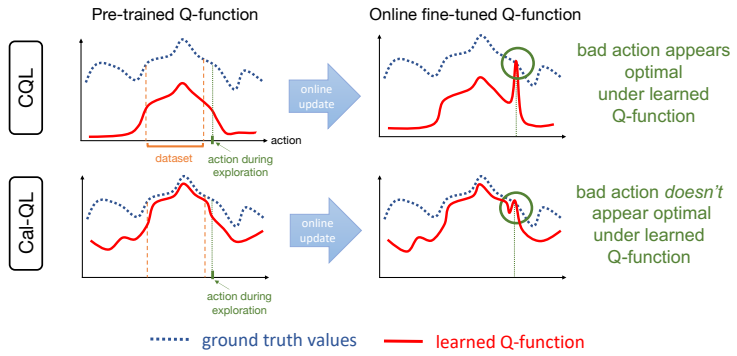


Figure 33: Intuition behind policy unlearning with CQL and the idea behind Cal-QL. The plot visualizes a slice of the learned Q-function and the ground-truth values for a given state. Erroneous peaks on suboptimal actions (x-axis) arise when updating CQL Q-functions with online data. This in turn can lead the policy to deviate away from high-reward actions covered by the dataset in favor of erroneous new actions, resulting in deterioration of the pre-trained policy. In contrast, Cal-QL corrects the scale of the learned Q-values by using a reference value function, such that actions with worse Q-values than the reference value function do not erroneously appear optimal in fine-tuning.

8.6 THEORETICAL INTUITION OF CAL-QL

We will now analyze the cumulative regret attained over online fine-tuning, when the value function is pre-trained with Cal-QL, and show that enforcing calibration (Defintion 8.4.1) leads to a favorable regret bound during the online phase. Our analysis utilizes tools from Song et al. [308], but studies the impact of calibration on fine-tuning. We also remark that we simplify the treatment of certain aspects (e.g., how to incorporate pessimism) as it allows us to cleanly demonstrate benefits of calibration.

Notation & terminology. In our analysis, we will consider an idealized version of Cal-QL for simplicity. Specifically, following prior work [308] under the bilinear model [66], we will operate in a finite-horizon setting with a horizon H. We denote the learned

Q-function at each learning iteration k for a given $(\boldsymbol{s}, \boldsymbol{a})$ pair and time-step h by $Q^k_\theta(\boldsymbol{s}, \boldsymbol{a})$. For any given policy π, let $C_\pi \geq 1$ denote the concentrability coefficient such that $C_\pi := \max_{f \in \mathcal{C}} \frac{\sum_{h=0}^{H-1} \mathbb{E}_{s,a \sim d^\pi_h}[\mathcal{T} f_{h+1}(s,a) - f_h(s,a)]}{\sqrt{\sum_{h=0}^{H-1} \mathbb{E}_{s,a \sim \nu_h}(\mathcal{T} f_{h+1}(s,a) - f_h(s,a))^2}}$, i.e., a coefficient that quantifies the distribution shift between the policy π and the dataset $\mathcal{D}$, in terms of the ratio of Bellman errors averaged under π and the dataset $\mathcal{D}$. Note that $\mathcal{C}$ represents the Q-function class and we assume $\mathcal{C}$ has a bellman-bilinear rank [66] of d. We also use C^μ_π to denote the concentrability coefficient over a subset of *calibrated* Q-functions w.r.t. a reference policy μ: $C^\mu_\pi := \max_{f \in \mathcal{C}, f(s,a) \geq Q^\mu(s,a)} \frac{\sum_{h=0}^{H-1} \mathbb{E}_{s,a \sim d^\pi_h}[\mathcal{T} f_{h+1}(s,a) - f_h(s,a)]}{\sqrt{\sum_{h=0}^{H-1} \mathbb{E}_{s,a \sim \nu_h}(\mathcal{T} f_{h+1}(s,a) - f_h(s,a))^2}}$, which provides $C^\mu_\pi \leq C_\pi$. Similar to $\mathcal{C}$, let d_μ denote the bellman bilinear rank of $\mathcal{C}_\mu$ – the calibrated Q-function class w.r.t. the reference policy μ. Intuitively, we have $\mathcal{C}_\mu \subset \mathcal{C}$, which implies that $d_\mu \leq d$. We will use π^k to denote the arg-max policy induced by Q^k_θ.

Intuition. We intuitively discuss how calibration and conservatism enable Cal-QL to attain a smaller regret compared to not imposing calibration. Our goal is to bound the cumulative regret of online fine-tuning, $\sum_k \mathbb{E}_{\boldsymbol{s}_0 \sim \rho}[V^{\pi^\star}(\boldsymbol{s}_0) - V^{\pi^k}(\boldsymbol{s}_0)]$. We can decompose this expression into two terms:

$$\mathrm{Reg}(K) = \underbrace{\sum_{k=1}^{K} \mathbb{E}_{\boldsymbol{s}_0 \sim \rho}\left[V^\star(\boldsymbol{s}_0) - \max_a Q^k_\theta(\boldsymbol{s}_0, \boldsymbol{a})\right]}_{(i) := \text{ miscalibration}} + \underbrace{\sum_{k=1}^{K} \mathbb{E}_{\boldsymbol{s}_0 \sim \rho}\left[\max_a Q^k_\theta(\boldsymbol{s}_0, \boldsymbol{a}) - V^{\pi^k}(\boldsymbol{s}_0)\right]}_{(ii) := \text{ overestimation}}. \quad (8.6.1)$$

This decomposition of regret into terms (i) and (ii) is instructive. Term (ii) corresponds to the amount of over-estimation in the learned value function, which is expected to be small if a conservative RL algorithm is used for training. Term (i) is the difference between the ground-truth value of the optimal policy and the learned Q-function and is negative if the learned Q-function were calibrated against the optimal policy (per Definition 8.4.1). Of course, this is not always possible because we do not know $V^\star$ a priori. But note that when Cal-QL utilizes a reference policy μ with a high value V^μ, close to $V^\star$, then the learned Q-function Q_θ is calibrated with respect to Q^μ per Condition 8.4.1 and term (i) can still be controlled. Therefore, controlling this regret requires striking a balance between learning a calibrated (term (i)) and conservative (term (ii)) Q-function. We now formalize this intuition and defer the detailed proof to Appendix F.2.5.

Theorem 8.6.1 (Informal regret bound of Cal-QL)**.** *With high probability, Cal-QL obtains the following bound on total regret accumulated during online fine-tuning:*

$$\mathrm{Reg}(K) = \widetilde{O}\Big(\min\left\{ C^\mu_{\pi^\star} H \sqrt{dK \log(|\mathcal{F}|)},\ K\mathbb{E}_\rho[V^\star(\boldsymbol{s}_0) - V^\mu(\boldsymbol{s}_0)] + H\sqrt{d_\mu K \log(|\mathcal{F}|)} \right\} \Big)$$

where $\mathcal{F}$ is the functional class of the Q-function.

Comparison to Song et al. [308]. Song et al. [308] analyzes an online RL algorithm that utilizes offline data without imposing conservatism or calibration. We now compare Theorem 8.6.1 to Theorem 1 of Song et al. [308] to understand the impact of these conditions on the final regret guarantee. Theorem 1 of Song et al. [308] presents a regret bound: $\mathrm{Reg}(K) = \widetilde{O}\left(C_{\pi^\star} H \sqrt{dK \log\left(|\mathcal{F}|\right)}\right)$ and we note some improvements in our guarantee, that we also verify via experiments in Section 8.7.3: **(a)** for the setting where the reference policy μ contains near-optimal behavior, i.e., $V^\star - V^\mu \lesssim O(H\sqrt{d \log\left(|\mathcal{F}|\right)/K})$, Cal-QL can enable a tighter regret guarantee compared to Song et al. [308]; **(b)** as we show in Appendix F.2.2, the concentrability coefficient $C^\mu_{\pi^\star}$ appearing in our guarantee is no larger than the one that appears in Theorem 1 of Song et al. [308], providing another source of improvement; and **(c)** finally, in the case where the reference policy has broad coverage *and* is highly sub-optimal, Cal-QL reverts back to the guarantee from [308], meaning that Cal-QL improves upon this prior work.

8.7 Experimental Evaluation

The goal of our experimental evaluation is to study how well Cal-QL can facilitate sample-efficient online fine-tuning. To this end, we compare Cal-QL with several other state-of-the-art fine-tuning methods on a variety of offline RL benchmark tasks from D4RL [92], Singh et al. [303], and Nair et al. [241], evaluating performance before and after fine-tuning. We also study the effectiveness of Cal-QL on higher-dimensional tasks, where the policy and value function must process raw image observations. Finally, we perform empirical studies to understand the efficacy of Cal-QL with different dataset compositions and the impact of errors in the reference value function estimation.

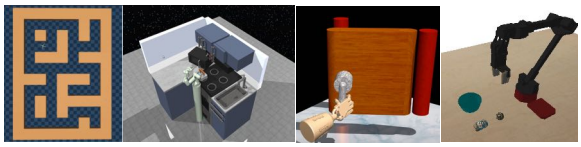


Figure 34: Tasks: We evaluate Cal-QL on a diverse set of benchmarks: AntMaze & Frankakitchen domains from [92], Adroit tasks from [241] and a vision-based robotic manipulation task from [188].

Offline RL tasks and datasets. We evaluate Cal-QL on a number of benchmark tasks and datasets used by prior works [175, 241] to evaluate fine-tuning performance: **(1)** The AntMaze tasks from D4RL [92] that require controlling an ant quadruped robot to navigate from a starting point to a desired goal location in a maze. The reward is +1 if the agent reaches within a pre-specified small radius around the goal and 0 otherwise. **(2)** The FrankaKitchen tasks from D4RL require controlling a 9-DoF Franka robot to attain a desired configuration of a kitchen. To succeed, a policy must complete four sub-tasks in the kitchen within a single rollout, and it receives a binary reward of +1/0 for every sub-task it completes. **(3)** Three Adroit dexterous manipulation tasks [274, 175, 241] that require learning complex manipulation skills on a 28-DoF five-fingered hand to **(a)** manipulate a pen in-hand to a desired configuration (pen-binary), **(b)** open a door by unlatching the handle (door-binary), and **(c)** relocating a ball to a desired location (relocate-binary). The agent obtains a sparse binary +1/0 reward

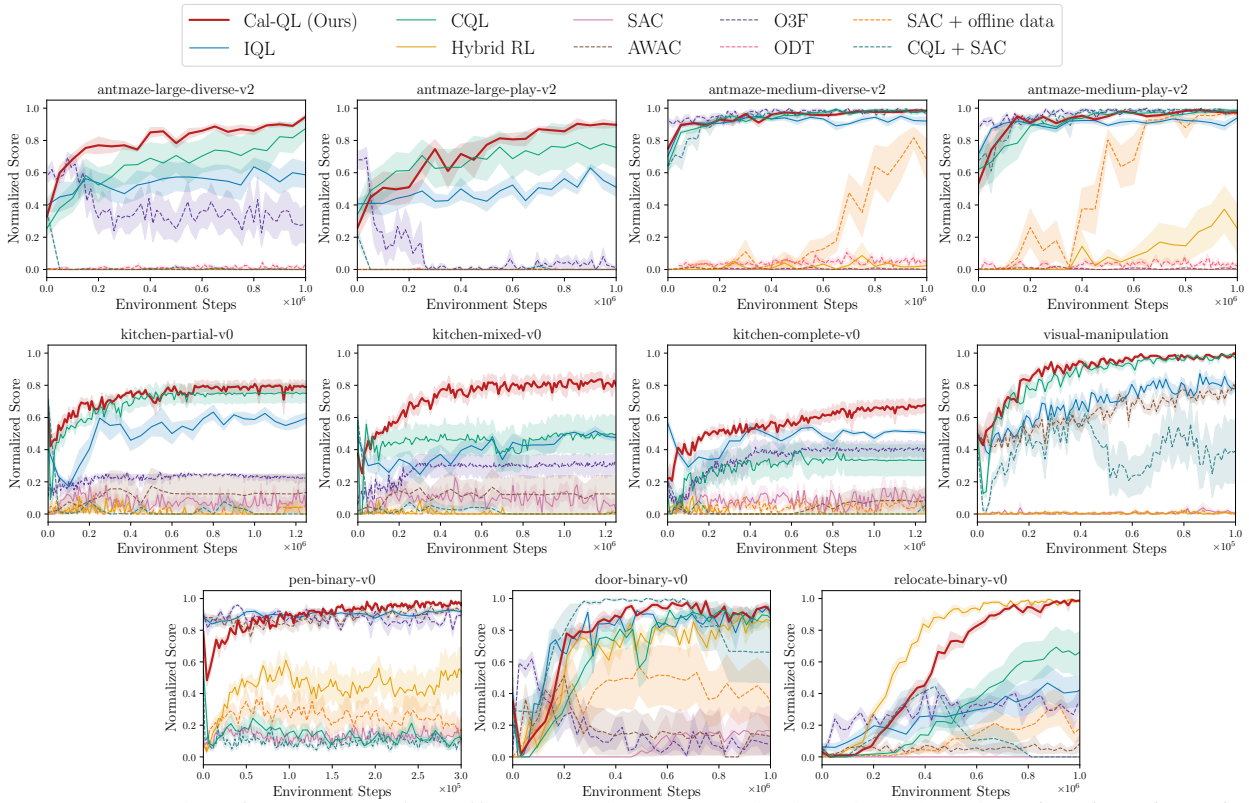


Figure 35: Online fine-tuning after offline initialization on the benchmark tasks. The plots show the online fine-tuning phase *after* pre-training for each method (except SAC-based approaches which are not pre-trained). Observe that Cal-QL consistently matches or exceeds the speed and final performance of the best prior method and is the only algorithm to do so across all tasks. (6 seeds)

if it succeeds in solving the task. Each of these tasks only provides a narrow offline dataset consisting of 25 demonstrations collected via human teleoperation and additional trajectories collected by a BC policy. Finally, to evaluate the efficacy of Cal-QL on a task where we learn from raw visual observations, we study **(4)** a pick-and-place task from prior work [303, 188] that requires learning to pick a ball and place it in a bowl, in the presence of distractors.

Comparisons, prior methods, and evaluation protocol. We compare Cal-QL to running online SAC [126] from scratch, as well as prior approaches that leverage offline data. This includes naïvely fine-tuning offline RL methods such as CQL [181] and IQL [175], as well as fine-tuning with AWAC [241], O3F [228] and online decision transformer (ODT) [374], methods specifically designed for offline RL followed by online fine-tuning. In addition, we also compare to a baseline that trains SAC [126] using both online data and offline data (denoted by "SAC + offline data") that mimics DDPGfD [331] but utilizes SAC instead of DDPG. We also compare to Hybrid RL [308], a recently proposed method that improves the sample efficiency of the "SAC + offline data" approach, and "CQL+SAC", which first pre-train with CQL and then run fine-tuning with SAC on a mixture of offline and online

data without conservatism. More details of each method can be found in Appendix F.5. We present learning curves for online fine-tuning and also quantitatively evaluate each method on its ability to improve the initialization learned from offline data measured in terms of **(i)** final performance after a pre-defined number of steps per domain and **(ii)** the cumulative regret over the course of online fine-tuning. In Section 8.7.2, we run Cal-QL with a higher update-to-data (UTD) ratio and compare it to RLPD [20], a more sample-efficient version of "SAC + offline data".

8.7.1 *Empirical Results*

We first present a comparison of Cal-QL in terms of the normalized performance before and after fine-tuning in Table 20 and the cumulative regret in a fixed number of online steps in Table 21. Following the protocol of [92], we normalize the average return values for each domain with respect to the highest possible return (+4 in FrankaKitchen; +1 in other tasks; see Appendix F.5.1 for more details).

Cal-QL improves the offline initialization significantly. Observe in Table 20 and Figure 35 that while the performance of offline initialization acquired by Cal-QL is comparable to that of other methods such as CQL and IQL, Cal-QL is able to improve over its offline initialization the most by **106.9%** in aggregate and achieve the best fine-tuned performance in **9 out of 11** tasks.

Cal-QL enables fast fine-tuning. Observe in Table 21 that Cal-QL achieves the smallest regret on **8 out of 11** tasks, attaining an average regret of 0.22 which improves over the next best method (IQL) by **42%**. Intuitively, this means that Cal-QL does not require running highly sub-optimal policies. In tasks such as relocate-binary, Cal-QL enjoys the fast online learning benefits associated with naïve online RL methods that incorporate the offline data in the replay buffer (SAC + offline data and Cal-QL are the only two methods to attain a score of $\geq$ 90% on this task) unlike prior offline RL methods. As shown in Figure 35, in the kitchen and antmaze domains, Cal-QL brings the benefits of fast online learning together with a good offline initialization, improving drastically on the regret metric. Finally, observe that the initial unlearning at the beginning of fine-tuning with conservative methods observed in Section 8.4.1 is greatly alleviated in all tasks (see Appendix F.8 for details).

8.7.2 *Cal-QL With High Update-to-Data (UTD) Ratio*

We can further enhance the online sample efficiency of Cal-QL by increasing the number of gradient steps per environment step made by the algorithm. The number of updates per environment step is usually called the update-to-data (UTD) ratio. In standard online RL, running off-policy Q-learning with a high UTD value (e.g., 20, compared to the typical value of 1) often results in challenges pertaining to overfitting [209, 46, 20, 64].

Task	CQL	IQL	AWAC	O3F	ODT	CQL+SAC	Hybrid SRL	SAC+od	SAC	Cal-QL (Ours)
large-diverse	25 → 87	40 → 59	00 → 00	59 → 28	00 → 01	36 → 00	→ 00	→ 00	→ 00	33 → **95**
large-play	34 → 76	41 → 51	00 → 00	68 → 01	00 → 00	21 → 00	→ 00	→ 00	→ 00	26 → **90**
medium-diverse	65 → **98**	70 → 92	00 → 00	92 → 97	00 → 03	64 → **98**	→ 02	→ 68	→ 00	75 → **98**
medium-play	62 → 98	72 → 94	00 → 00	89 → **99**	00 → 05	67 → 98	→ 25	→ 96	→ 00	54 → 97
partial	71 → 75	40 → 60	01 → 13	11 → 22	-	71 → 00	→ 00	→ 07	→ 03	67 → **79**
mixed	56 → 50	48 → 48	02 → 12	06 → 33	-	59 → 01	→ 01	→ 00	→ 02	38 → **80**
complete	13 → 34	57 → 50	01 → 08	17 → 41	-	21 → 06	→ 00	→ 05	→ 06	22 → **68**
pen	55 → 13	88 → 92	88 → 92	91 → 89	-	48 → 10	→ 54	→ 17	→ 11	79 → **99**
door	22 → 88	41 → 88	29 → 13	04 → 08	-	29 → 66	→ 88	→ 39	→ 17	35 → **92**
relocate	06 → 69	06 → 45	06 → 08	03 → 35	-	01 → 00	→ **99**	→ 16	→ 00	03 → 98
manipulation	50 → 97	49 → 81	50 → 73	-	-	42 → 41	→ 00	→ 01	→ 01	49 → **99**
average	42 → 71	50 → 69	16 → 20	44 → 45	00 → 02	42 → 29	→ 24	→ 23	→ 04	44 → **90**
improvement	+ 71.0%	+ 37.7%	+ 23.7%	+ 3.0%	N/A [a]	- 30.3%	N/A	N/A	N/A	**+ 106.9%**

Table 20: Normalized score before & after online fine-tuning. Observe that Cal-QL improves over the best prior fine-tuning method and attains a much larger performance improvement over the course of online fine-tuning. The numbers represent the normalized score out of 100 following the convention in [92].

Task	CQL	IQL	AWAC	O3F	ODT	CQL+SAC	Hybrid RL	SAC+od	SAC	Cal-QL (Ours)
large-diverse	0.35	0.46	1.00	0.62	0.98	0.99	1.00	1.00	1.00	**0.20**
large-play	0.32	0.52	1.00	0.91	1.00	0.99	1.00	1.00	1.00	**0.28**
medium-diverse	0.06	0.08	0.99	**0.03**	0.95	0.06	0.98	0.77	1.00	0.05
medium-play	0.09	0.10	0.99	**0.04**	0.96	0.06	0.90	0.47	1.00	0.07
partial	0.31	0.49	0.89	0.78	-	0.97	0.98	0.98	0.92	**0.27**
mixed	0.55	0.60	0.88	0.72	-	0.97	0.99	1.00	0.91	**0.27**
complete	0.70	0.53	0.97	0.66	-	0.99	0.99	0.96	0.91	**0.44**
pen	0.86	**0.11**	0.12	0.13	-	0.90	0.56	0.75	0.87	**0.11**
door	0.36	0.25	0.81	0.82	-	**0.23**	0.35	0.60	0.94	**0.23**
relocate	0.71	0.74	0.95	0.71	-	0.86	**0.30**	0.89	1.00	0.43
manipulation	0.15	0.32	0.38	-	-	0.61	1.00	1.00	0.99	**0.11**
average	0.41	0.38	0.82	0.54	0.97	0.69	0.82	0.86	0.96	**0.22**

Table 21: Cumulative regret averaged over the steps of fine-tuning. The smaller the better and 1.00 is the worst. Cal-QL attains the smallest overall regret, achieving the best performance among 8 / 11 tasks.

As expected, we noticed that running Cal-QL with a high UTD value also leads these overfitting challenges. To address these challenges in high UTD settings, we combine Cal-QL with the Q-function architecture in recent work, RLPD [20] (i.e., we utilized layer normalization in the Q-function and ensembles akin to [46]), that attempts to tackle overfitting challenges. Note that Cal-QL still first pre-trains on the offline dataset using Equation 8.5.1 followed by online fine-tuning, unlike RLPD that runs online RL right from the start. In Figure 36, we compare Cal-QL (UTD = 20) with RLPD [20] (UTD = 20) and also Cal-QL (UTD = 1) as a baseline. Observe that Cal-QL (UTD = 20) improves over Cal-QL (UTD = 1) and training from scratch (RLPD).

8.7.3 *Understanding the Behavior of Cal-QL*

In this section, we aim to understand the behavior of Cal-QL by performing controlled experiments that modify the dataset composition, and by investigating various metrics to understand the properties of scenarios where utilizing Cal-QL is especially important for online fine-tuning.

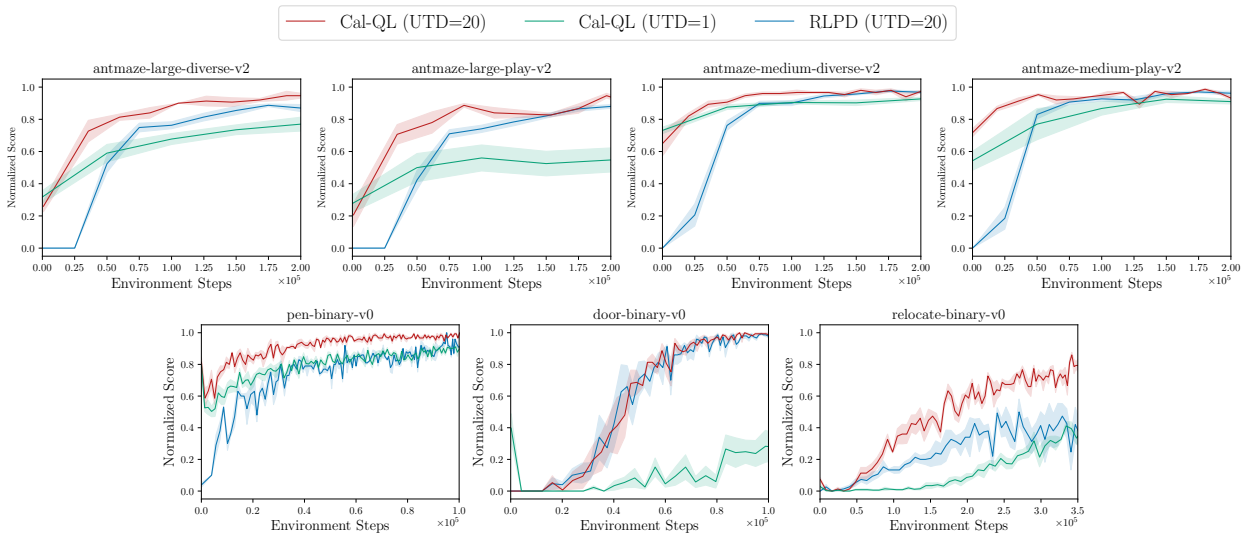


Figure 36: Cal-QL with UTD=20. Incorporating design choices from RLPD enables Cal-QL to achieve sample-efficient fine-tuning with UTD=20. Specifically, Cal-QL generally attains similar or higher asymptotic performance as RLPD, while also exhibiting a smaller cumulative regret.

Effect of data composition. To understand the efficacy of Cal-QL with different data compositions, we ran it on a newly constructed fine-tuning task on the medium-size AntMaze domain with a low-coverage offline dataset, which is generated via a scripted controller that starts from a fixed initial position and navigates the ant to a fixed goal position. In Figure 37, we plot the performance of Cal-QL and baseline CQL (for comparison) on this task, alongside the trend of average Q-values over the course of offline pre-training (to the left of the dashed vertical line, before 250 training epochs) and online fine-tuning (to the right of the vertical dashed line, after 250 training epochs), and the trend of *bounding rate*, i.e., the fraction of transitions in the data-buffer for which the constraint in Cal-QL actively lower-bounds the learned Q-function with the reference value. For comparison, we also plot these quantities for a diverse dataset with high coverage on the task (we use the antmaze-medium-diverse from [92] as a representative diverse dataset) in Figure 37.

Observe that for the diverse dataset, both naïve CQL and Cal-QL perform similarly, and indeed, the learned Q-values behave similarly for both of these methods. In this setting, online learning doesn't spend samples to correct the Q-function when fine-tuning begins leading to a low bounding rate, almost always close to 0. Instead, with the narrow dataset, we observe that the Q-values learned by naïve CQL are much smaller, and are corrected once fine-tuning begins. This correction co-occurs with a drop in performance (solid blue line on left), and naïve CQL is unable to recover from this drop. Cal-QL which calibrates the scale of the Q-function for many more samples in the dataset, stably transitions to online fine-tuning with no unlearning (solid red line on left).

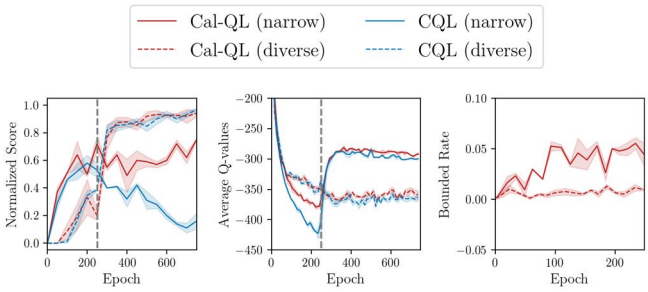


Figure 37: Performance of Cal-QL with data compositions. Cal-QL is most effective with narrow datasets, where Q-values need to be corrected at the beginning of fine-tuning.

This suggests that in settings with narrow datasets (e.g., in the experiment above and in the adroit and visual-manipulation domains from Figure 35), Q-values learned by naïve conservative methods are more likely to be smaller than the ground-truth Q-function of the behavior policy due to function approximation errors. Hence utilizing Cal-QL to calibrate the Q-function against the behavior policy can be significantly helpful. On the other hand, with significantly high-coverage datasets, especially in problems where the behavior policy is also random and sub-optimal, Q-values learned by naïve methods are likely to already be calibrated with respect to those of the behavior policy. Therefore no explicit calibration might be needed (and indeed, the bounding rate tends to be very close to 0 as shown in Figure 37). In this case, Cal-QL will revert back to standard CQL, as we observe in the case of the diverse dataset above. This intuition is also reflected in Theorem 8.6.1: when the reference policy μ is close to a narrow, expert policy, we would expect Cal-QL to be especially effective in controlling the efficiency of online fine-tuning. We also present a diagnostic study of Cal-QL when the reference value function is estimated by fitting a neural network in Appendix F.7, and find that estimation errors in this model of the reference function do not affect performance significantly.

8.8 DISCUSSION AND LIMITATIONS

In this chapter, we developed Cal-QL a method for acquiring conservative offline initializations that facilitate fast online fine-tuning. Cal-QL learns conservative value functions that are constrained to be larger than the value function of a reference policy. This form of calibration allows us to avoid initial unlearning when fine-tuning with conservative methods, while also retaining the effective asymptotic performance that these methods exhibit. Our theoretical and experimental results highlight the benefit of Cal-QL in enabling fast online fine-tuning. While Cal-QL outperforms prior methods, we believe that we can develop even more effective methods by adjusting calibration and conservatism more carefully. A limitation of our work is that we do not consider fine-tuning setups where pre-training and fine-tuning tasks are different, but this is an interesting avenue for future work. Theoretically, our analysis can be improved via a more precise treatment of pessimism, and this is an avenue for future work.

Part IV

APPLICATIONS OF OFFLINE RL

9

REAL-ROBOT PRE-TRAINING

Abstract

In this chapter, we present an application of offline RL that leverages diverse robotic datasets to achieve effective generalization in robotic learning. Specifically, we aim to answer the question: How can we use existing diverse offline datasets along with small amounts of task-specific data to solve new tasks? We demonstrate that employing the end-to-end offline RL techniques discussed in earlier sections of this dissertation can be an effective approach to address this, without the need for any self-supervised representation learning or vision-based pre-training. Our offline RL approach, "pre-training for robots" (PTR), aims to efficiently learn new tasks by combining pre-training on existing robotic datasets with rapid fine-tuning on a new task, with as few as 10 demonstrations. PTR directly utilizes conservative Q-learning (CQL) from Chapter 5, but extends it with several crucial design decisions that enable PTR to outperform various prior methods. To the best of our knowledge, PTR represents the first RL method that successfully learns new tasks in a new domain on a real WidowX robot, with limited data, effectively leveraging an existing dataset of diverse multi-task robot data collected from various toy kitchens. Furthermore, we demonstrate that PTR enables effective autonomous online improvement in just a handful of trials.

9.1 INTRODUCTION

In this chapter, we show that multi-task offline RL pre-training on diverse multi-task demonstration data followed by offline RL fine-tuning on a very small number of trajectories (as few as 10 trials, maximum 15) or online fine-tuning on autonomously collected data, can indeed be made into an effective robotic learning strategy that can significantly outperform methods based on imitation learning as well as RL-based methods that do not employ pre-training. This is surprising and significant, since prior work [227] has suggested that imitation learning methods are superior to offline RL when provided

with human demonstrations. Moreover prior RL-based pre-training and fine-tuning methods typically require thousands of trials [303, 162, 154, 41, 197]. Our framework, which we call PTR (pre-training for robots), is based on the CQL algorithm discussed previously, but introduces a number of design decisions, that we show are critical for good performance and enable large-scale pre-training. These choices include a specific choice of architecture for providing high capacity while preserving spatial information, the use of group normalization, and an approach for feeding actions into the model that ensures that actions are used properly for value prediction. We experimentally validate these design decisions and show that PTR benefits from increasing the network capacity, even with large ResNet-50 architectures, which have never been previously shown to work with offline RL. Our experiments utilize the Bridge Dataset [70], which is an extensive dataset consisting of thousands of trials for a very large number of robotic manipulation tasks in multiple environments. A schematic of PTR is shown in Figure 38.

9.2 PROBLEM STATEMENT AND DEFINITIONS

Problem statement. Our goal is to learn general-purpose initializations from a broad, multi-task offline dataset and then fine-tune these initializations to specific downstream tasks. Following the notation from Chapter 7, we denote the general-purpose offline dataset by $\mathcal{D}$, which is partitioned into k chunks. Each chunk contains several transition typles for a given robotic task (e.g., picking and placing a given object) collected in a given domain (e.g., a particular kitchen). See Figure 38 for an illustration. Formally, the dataset can be represented as $\mathcal{D} = \cup_{i=1}^{k} (i, \mathcal{D}_i)$, where we denote the set of training tasks concisely as $\mathcal{T}_{\text{train}} = [k]$. Our goal is to utilize this multi-task dataset to help train a policy for one or multiple target tasks (denoted without loss of generality as task $\mathcal{T}_{\text{target}} = \{k+1, \cdots, n\}$).

While the diverse prior dataset $\mathcal{D}$ does not contain any experience for the target tasks, in the offline fine-tuning setting, we are provided with a very small dataset of demonstrations $\mathcal{D}^* := \{\mathcal{D}^*_{k+1}, \mathcal{D}^*_{k+1}, \cdots, \mathcal{D}^*_n\}$ corresponding to each of the target tasks. In our experiments, we use only 10 to 15 demonstrations for each target task, making it impossible to learn the target task from this data alone, such that a method that effectively maximizes performance for the target tasks $\mathcal{T}_{\text{target}}$ must leverage the prior data $\mathcal{D}$. We also study the setting where we aim to quickly fine-tune the policy learned via offline pre-training and offline fine-tuning using limited amounts of autonomously collected data via online real-world interaction. More details about this are in Section 9.4.6.

Tasks and domains. We use the Bridge Dataset [70] as the source of our pre-training tasks, which we augment with a few additional tasks as discussed in Section 9.4. Our terminology for "task" and "domain" follows Ebert et al. [70]: a task is a skill-object pair, such as "put potato in pot" and a domain corresponds to an environment, which in the

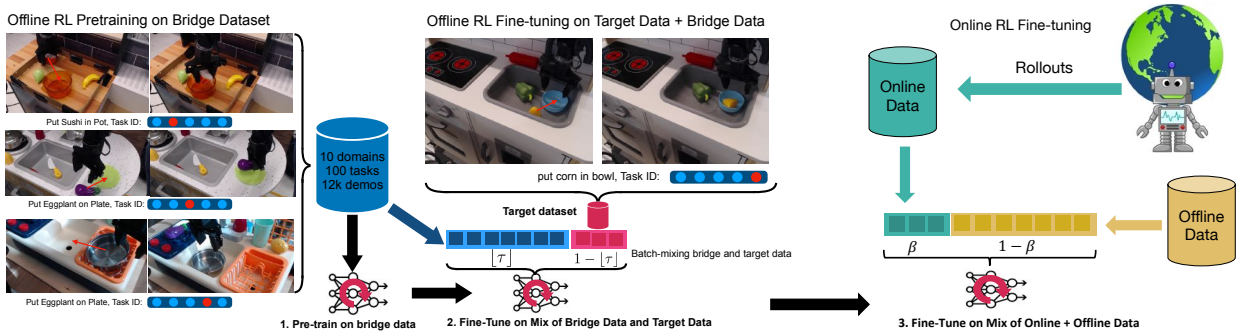


Figure 38: Overview of PTR : We first perform general offline pre-training on diverse multi-task robot data and subsequently fine-tune on one or several target tasks while mixing batches between the prior data and the target dataset using a batch mixing ratio of τ. Additionally, a separate online fine-tuning phase can be done, where offline pre-training is done on a static dataset and an online replay buffer is collected using rollouts in the environment. The offline and online buffers are mixed per batch with a ratio of β.

case of the Bridge Dataset consists of different toy kitchens, potentially with different viewpoints and robot placements. We assume the new tasks and environments come from the same training distribution, but are not seen in the prior data.

9.3 LEARNING POLICIES FOR NEW TASKS FROM OFFLINE RL PRE-TRAINING

To effectively solve new tasks from diverse offline datasets, a robotic learning framework must: **(1)** extract useful skills out of the diverse robotic dataset, and **(2)** rapidly specialize the learned skills towards an unseen target task, given only a minimal amount of experience from this target task in the form of demonstrations, or collected autonomously by interaction. In this section, we present our framework, PTR , that provides these benefits by training a single, highly expressive deep network via offline RL, and then specializes it on the target task with a small amount of data. We will first present the key components of our robotic framework in Section 9.3.1 and then discuss our novel technical contributions, the practical design choices that are crucial, in Section 9.3.2.

9.3.1 *The Components of PTR*

To satisfy both requirements **(1)** and **(2)** from above, our framework uses a multi-task offline RL approach with parameter sharing, where the policy and Q-function are conditioned on a task identifier. This allows us to share a single set of weights for all possible tasks in the diverse offline dataset, providing a general-purpose pre-training procedure that can use diverse data. Once a policy is obtained via this multi-task pre-training process, we adapt this policy for solving a new target task by utilizing a very small amount of target task data or autonomously collected data. We describe the two phases, pre-training and fine-tuning, below:

Phase 1: Multi-task offline RL pre-training. In the first phase, PTR learns a single Q-function and policy for all tasks $i \in \mathcal{T}_{\text{train}}$ conditioned on the task identifier i, i.e., $Q_\phi(\boldsymbol{s}, \boldsymbol{a}; i)$ and $\pi_\theta(\boldsymbol{a}|\boldsymbol{s}, i)$, via multi-task offline RL. We use a one-hot task identifier that imposes minimal assumptions on the task structure. For multi-task offline RL, we use the conservative Q-learning (CQL) [181] algorithm. Recall that this amounts to training the multi-task Q-function against a temporal difference error objective along with a regularizer that explicitly minimizes the expected Q-value under the learned policy $\pi_\theta(\boldsymbol{a}|\boldsymbol{s}; i)$, to prevent overestimation of Q-values for unseen actions, which can lead to poor offline RL performance [179]. At the end of multi-task offline training phase, we obtain a policy π_θ^{off} and Q-function Q_ϕ^{off}, that are ready to be fine-tuned to a new downstream task.

Phase 2: Offline or online fine-tuning of π_θ^{off} and Q_ϕ^{off} to a target task $\mathcal{T}_{\text{target}}$. In the second phase, PTR attempts to learn a policy to solve one or more downstream tasks by adapting π_θ^{off}, using a limited set of user-provided demonstrations that we denote $\mathcal{D}^*$, or using a combination of target demonstration data and autonomously collected online data. Our method for the offline fine-tuning setting is simple yet effective: we incorporate the new target task data into the replay buffer of the very same offline multi-task CQL algorithm from the previous phase and resume training from Phase 1. However, naïvely incorporating the target task data into the replay buffer might still not be effective since this scheme would hardly ever train on the target task data during adaptation due to the large imbalance between the sizes of the few target demonstrations and the large pre-training dataset. To address this imbalance, each minibatch passed to multi-task CQL during offline fine-tuning consists of a τ fraction of transitions from bridge demonstration data and $1 - \tau$ fraction of transitions from the target dataset. By setting τ to be small, we are able to prioritize multi-task CQL to look at target task data frequently, enabling it to make progress on the downstream task without overfitting.

For the autonomous fine-tuning, we utilize a similar technique and have each mini-batch consist of β fraction of transitions from the bridge data and the target demonstration data, and $1 - \beta$ fraction of transitions from the newly collected online data. We alternate between collecting one trajectory and making 10 gradient steps for every single transition collected in the environment. Utilizing a high update to the data ratio allowed us to efficiently train the agent on newly collected online samples from rollouts.

Handling task identifiers for new tasks. The description of our system so far has assumed that the downstream test tasks are identified via a task identifier. In practice, we utilize a one-hot vector to indicate the index of a task. While such a scheme is simple to implement, it is not quite obvious how we should incorporate new tasks with one-hot task identifiers. In our experiments, we use two approaches for solving this problem: first, we can utilize a larger one-hot encoding that incorporates tasks in both $\mathcal{T}_{\text{train}}$ and $\mathcal{T}_{\text{target}}$,

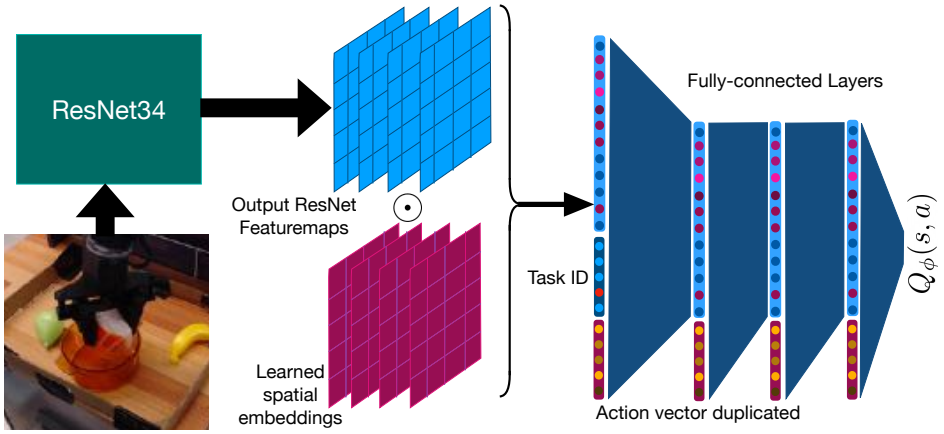


Figure 2: Q-function architecture for PTR . The encoder is a ResNet34 with group normalization along with learned spatial embeddings (left). The decoder (right) is a MLP with the action vector duplicated and passed in at each layer. A one-hot task identifier is also passed into the input of the decoder.

but not use the indices for $\mathcal{T}_{\text{target}}$ during pre-training. The Q-function and the policy are trained on these *placeholder* task identifiers only during fine-tuning in Phase 2. Another approach for handling new tasks is to not use unique task identifiers for every new task, but rather "*re-target*" or re-purpose existing task identifiers for new target tasks in the fine-tuning phase. PTR provides this option: we can simply assign an already existing task identifier to the target demonstration data before fine-tuning the learned Q-function and the policy. For example, in our experiments in Section 9.4 we re-target the put sushi in pot task which uses orange transparent pots to instead put the sushi into a metal pot, which was never seen during training.

An overview of our approach is shown in Figure 38. We use a value of $\alpha = 10.0$ and $\tau = 0.8$ for mixing the pre-training dataset and the target dataset in most of our experiments in the real-world, without requiring any domain-specific tuning. For online fine-tuning, we utilized $\alpha = 0.5$ to evenly mix between the online and offline datasets.

9.3.2 *Important Design Choices and Practical Considerations*

Even though the components discussed in Section 9.3.1 are sufficient to give rise to an offline pre-training and fine-tuning approach, as we show in Section 5, this approach does not lead very good results on its own. Instead, we must make some crucial design decisions, including designing neural network architectures that can learn from diverse data with offline RL, cross-validation metrics to identify policies we expect to be effective after fine-tuning, and the design of the reward functions that can be used to label the pre-training dataset. We show that making the right choices for these components leads to significant improvement (more than **3.5x** in final real-world performance; see Appendix G.5). Thus, describing, analyzing, and evaluating these choices is a crucial part of this work that we hope will facilitate applications of offline RL pre-training.

Policy and Q-function architectures

Perhaps the most crucial design decision for our approach is the neural network architecture for representing π^{off} and Q^{off}. Since we wish to fine-tune the policy for different tasks, we must use high-capacity neural network models for representing the policy and the Q-function. We experimented with a variety of standard (high-capacity) architectures for vision-based robotic RL. This includes standard convolutional architectures [303] and IMPALA architectures [74]. However, we observed in Figure 7 that these standard models were unable to effectively handle the diversity of the pre-training data and performed poorly. Then, we attempted to utilize standard ResNets [131] (ResNet-18, Resnet-34, and their adaptations to imitation problems from Ebert et al. [70]) to represent Q_ϕ, but faced divergence challenges similar to prior efforts that use batch normalization [30, 29] in the Q-network. Batch normalization layers are known to be hard to train with TD-learning [29] and, therefore, by replacing batch normalization layers with **group normalization** layers [344], we were able to address such divergence issues. See Appendix G.5 for quantitative studies comparing these choices. Unlike prior work [199], we observed that with group normalization, we attain favorable scaling properties of PTR : the more the parameters, the better the performance as shown in Figure 7. We also observed that choosing an appropriate method for converting the three-dimensional feature-map tensor produced by the ResNet into a one-dimensional embedding plays a crucial role for learning accurate Q-functions and obtaining functioning policies. Unlike standard ResNet architectures for supervised learning, simply utilizing global average pooling (as used in many classification architectures) performs poorly. Instead we pointwise multiply the learned feature-map with a 3-dimensional parameter tensor before computing sums over the spatial dimensions which allows the network to explicitly encode spatial information. We refer to this technique as "**learned spatial embeddings**". An illustration of this architecture is provided in Figure 2. As detailed in Appendix G.5, Table 67, we find that utilizing this technique leads to improved performance.

Next, we found that a Q-function $Q_\phi(\boldsymbol{s}, \boldsymbol{a})$ obtained by running naïve multi-task CQL on the demonstration data tends to not use the action input $\boldsymbol{a}$ effectively, due to strong correlations between $\boldsymbol{s}$ and $\boldsymbol{a}$ in the data, which is almost always the case for narrow, human demonstrations. As a result, policy improvement against such a Q-function overfits to these correlations, producing poor policies. To resolve this issue, we modified the architecture of Q-network to **pass the action a as input to every fully-connected layer** which, as shown in Figure 2 and Appendix G.5, Table 68), greatly alleviates the issue and significantly improves over naïve CQL.

Cross-validation during offline fine-tuning

As we wish to learn task-specific policies that do not overfit to small amounts of data, we must apply the right number of gradient steps during fine-tuning: too few gradient steps

will produce policies that do not succeed at the target tasks, while too many gradient steps will give policies that have likely lose the generalization ability of the pre-trained policy. To handle this trade-off, we adopt the following heuristic as a loose guideline: we run fine-tuning for many iterations while also plotting the learned Q-values over a held-out dataset of trajectories from the target task as seen in Figure 3. Then for evaluation, we pick the checkpoints that presented a Q-function with the Q-values appearing closest to having a monotonically increasing trend in a trajectory. This is a *relative* guideline and must be performed within the checkpoints observed within a run. The reason for this heuristic choice is that a valid Q-function must be a valid estimator for discounted return, and hence, it must increase over time-steps of a trajectory for a given task. Of course, this heuristic does not hold for arbitrary sub-optimal offline data, but all of our data comes from human-collected demonstrations. In principle, this heuristic can be wrapped into a metric quantifying degree of monotonicity of the Q-value curve in Figure 3, but in our experiments, we felt this was not necessary: as we show below, we were able to narrow down the checkpoints to essentially one or at most, two checkpoints by just visual inspection. Of course, designing an accurate metric would be helpful for future work. We present two worked-out examples of our checkpoint selection strategy for two tasks from Scenario 1 and Scenario 3 in Figure 3. Observe that checkpoints early in training exhibit Q-values that fluctuate arbitrarily at the beginning of training, which is clearly non-monotonic. This is because of the lack of sufficient gradient steps for fine-tuning the target task. Once sufficient gradient steps are performed, the Q-values visibly improve on the monotonicity property. Training further leads to much flatter Q-values, that are visibly less monotonic.

To validate this mechanism, in Figure 4 we present a film-strip of a sample evaluation of a good and a poor checkpoint as identified by the cross-validation strategy mentioned above. We observe that the checkpoint with more flat Q-values fails to solve the door opening task, whereas the one with a visibly increasing Q-value trend solves the task.

Reward specification

In this paper, we aim to pre-train on existing robotic datasets, such as the Bridge Dataset [70], which consists of human-teleoperated demonstration data. Although the demonstrations are all successful, they are not annotated with any reward function. Perhaps an obvious choice is to label the last transition of each trajectory as success, and give it a +1 binary reward. However, in several of the datasets we use, there can be a 0.5-1.0 second lag between task completion and when the episode is terminated by the data collection. To ensure that a successful transition is not incorrectly labeled as 0, we utilized the practical heuristic of annotating the last $n = 3$ transitions of every trajectory with a reward of $+1$ and and annotated other states with a 0 reward. We show in Appendix G.4 that this provided the best results. In principle, more complicated methods of reward

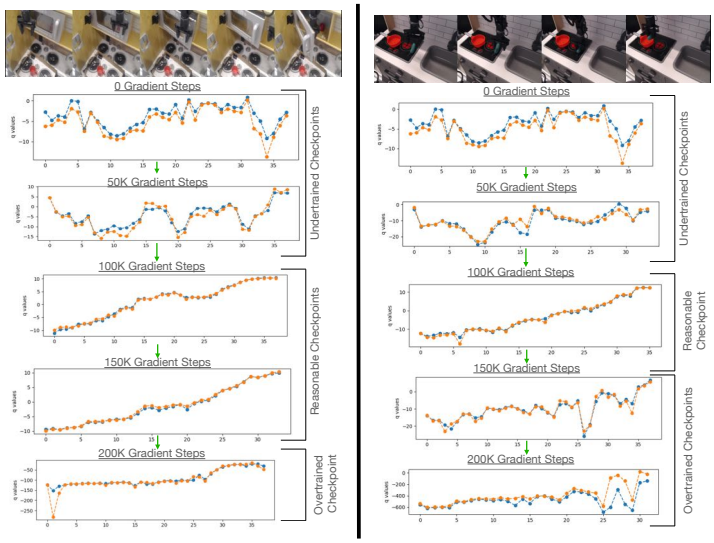


Figure 3: Evolution of Q-values on the target task over the process of fine-tuning with PTR . Observe that while the learned Q-values on *held-out* trajectories from the dataset just at the beginning of Phase 2 (fine-tuning) do not exhibit a roughly increasing trend, we choose to evaluate those checkpoints of PTR that exhibit a visible more increasing trend in the Q-values despite having access to only 10 demonstrations for these target tasks.

labeling [77] could be used. However, we found the presented rule to be simple and yet effective to learn good policies.

9.4 Experimental evaluation of PTR and takeaways for robotic RL

The goal of our experiments is to validate if PTR can learn effective policies from only a handful of user-provided demonstrations for a target task, by effectively utilizing previously-collected robotic datasets for pre-training. We also aim to understand whether the design decisions introduced in Section 9.3.2 are crucial for attaining good robotic manipulation performance. To this end, we evaluate PTR in a variety of robotic manipulation settings, and compare it to state of the art methods, which either do not use offline RL or do not learn end-to-end by employing some form of visual representation learning. We evaluate in three scenarios: **(a)** when the target task requires retargeting the behavior of an existing skill, in this case changing the type of object types it interacts with, **(b)** when the target task requires performing a previously observed task but this time in a previously unseen domain, and **(c)** when the target task requires learning a new skill in a new domain, by using the target demonstrations. We also perform a diagnostic study in simulation in Appendix G.1 (Table 62).

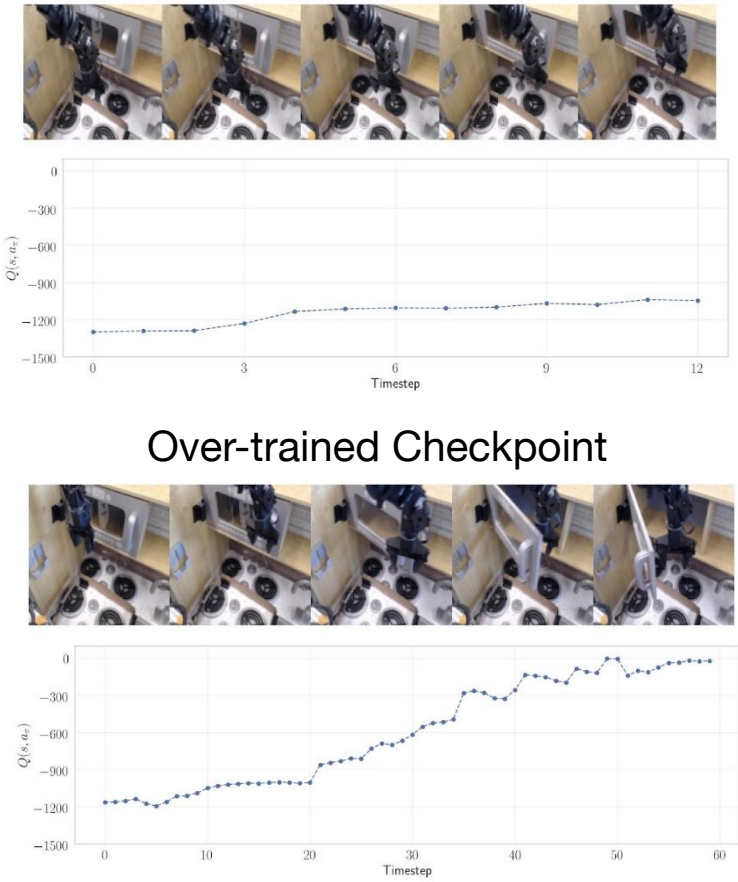


Figure 4: Performance evaluation of a selected and over-trained checkpoint of PTR . We validate our checkpoint selection mechanism on the door opening task. An over-trained checkpoint with nearly flat Q-values fails to solve the task, whereas a checkpoint with visibly increasing Q-values solves the task.

9.4.1 *Setup and Comparisons*

Real-world experimental setup. We directly utilize the publicly available *Bridge Dataset* [70] for pre-training, as it provides a large number of robot demonstrations for a diverse set of tasks in multiple domains, i.e., multiple different toy kitchens. We use the same WidowX250 robot platform for our evaluations. The bridge dataset contains distinct tasks, each differing in terms of the objects that the robot interacts with and the domain the task is situated in. We assign a different task identifier to each task in the dataset for pre-training. We also evaluate on an additional door-opening task not present in the Bridge Dataset, where we collected demonstrations for opening and closing a variety of doors, and test our system on new, unseen doors. More details are in Appendix G.2.

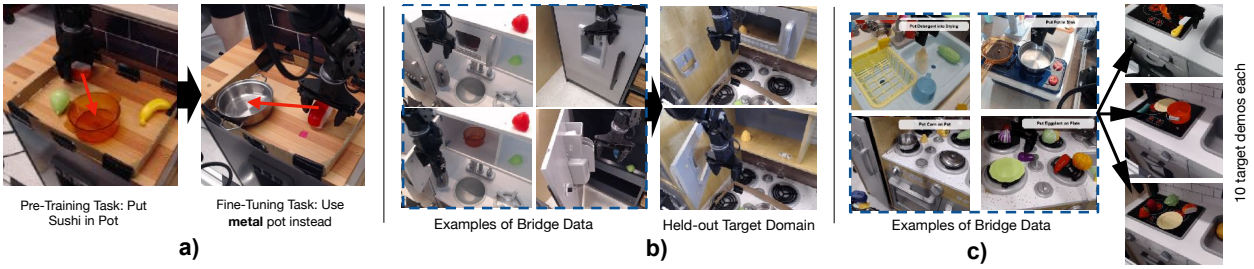


Figure 5: Illustrations of the three real-world experimental setups we evaluate PTR on: (a) the "put sushi in a metallic pot" task which requires retargeting, **(b)** the task of opening an unseen door, and **(c)** fine-tuning on several novel target tasks in a held out toykitchen environment.

Comparisons. Since the datasets we use (both the pre-training bridge dataset from [70] and the newly collected door opening data) consist of human demonstrations, as indicated by prior work [227], the strongest prior method in this setting is behavioral cloning (BC), which attempts to simply imitate the action of the demonstrator based on the current state. We incorporate BC in a pipeline similar to PTR , denoted as **BC (finetune)**, where we first run BC on the pre-training dataset, and then finetune it using the demonstrations on the target task using the same batch mixing as in PTR . To ensure that our BC baselines are well-tuned, we utilize standard practices of cross-validation via a held-out validation set to tune hyperparameters and make early stopping decisions are we elaborate on in Appendix G.4.1. Next, to assess the importance of performing pre-training *followed* by fine-tuning, we compare PTR to **(i)** jointly training on the pre-training and fine-tuning data with CQL, which is equivalent to the COG approach of Singh et al. [303], **(ii)** multi-task offline CQL (**CQL (zero-shot)**) that does not use the target demonstrations at all, and **(iii)** utilizing CQL to train on target demonstrations alone from scratch, with no pre-training data included (**CQL (target data only)**). We also make the analogous comparison for BC, jointly training BC on the pre-training and target task data from scratch (**BC (joint)**) which is equivalent to [70]. For fairness of comparison, BC, CQL, and PTR (both for zero-shot, joint-training and fine-tuning) use the *same* exact architecture, including our learned-spatial embedding described in Section 9.3.2.

9.4.2 *Experimental Results*

Method	Success rate
BC (zero-shot)	0/30
BC (finetune)	0/30
CQL (zero-shot)	2/30
PTR (Ours)	**14/30**

Table 22: Performance of PTR for "put sushi in metallic pot" in Scenario 1. PTR substantially outperforms BC (finetune), even though it is provided access to only demonstration data. We also show some examples comparing some trajectories of BC and PTR in Appendix G.4.

			zero-shot		Joint Training		Target data only	
Task	**PTR (Ours)**	**BC (fine.)**	**CQL**	**BC**	**COG**	**BC**	**CQL**	**BC**
Open Door	**12/20**	10/20	0/20	0/20	5/20	7/20	4/20	7/20

Table 23: Successes vs. total trials for opening a new target door in Scenario 2. PTR outperforms both BC (finetune) and BC (joint) given access to the same data. Note that joint training is worse than finetuning from the pre-trained initialization.

Scenario 1: Re-targeting skills for existing tasks

We utilized the subset of the bridge data with pick-and-place tasks in one toy kitchen for pre-training, and selected the "put sushi in pot" task as our target task. This task is depicted in the bridge dataset, but only using an orange transparent pot (see Figure 5 (a)). In order to construct a scenario where the offline policy at the end of pre-training must be re-targeted to act on a different object, we collected only *ten* demonstrations that place the sushi in a metallic pot and used these demonstrations for fine-tuning. This scenario is challenging since the metallic pot differs significantly from the orange transparent pot visually. By pre-training on all pick-and-place tasks in this domain (32 tasks) and fine-tuning on this data and 10 demonstrations, PTR is able to obtain a policy that is re-targeted towards the metal pot. On the other hand, the policy learned by BC confuses arbitrary patches on the tabletop with the pot. Quantitatively, observe in Table 22 that PTR is able to complete the task with reasonable accuracy across a set of easy and hard initial positions, whereas zero-shot and fine-tuned BC are completely unable to solve the task. The fact that zero-shot CQL has difficulty solving the task indicates that target demonstrations are necessary, and PTR is able to attain successful behavior with just ten demonstrations.

Scenario 2: Generalizing to previously unseen domains

Next, we study whether PTR can adapt behaviors seen in the pre-training data to new domains. We study a door opening task, which requires significantly more complex maneuvers and precise control compared to the pick-and-place tasks from above. The doors in the pre-training data exhibit different sizes, shapes, handle types and visual appearances, and the target door (shown in Figure 5(b)) we wish to open and the corresponding toy kitchen domain are never seen previously in the pre-training data. Concretely, for pre-training, we used a dataset of 800 door-opening demonstrations on 12 different doors in 4 different toy kitchen domains, and we utilize 15 demonstrations on a held-out door for fine-tuning. Table 23 shows that PTR improves over both BC baselines and joint training with CQL (or COG). Due to the limited target data and the associated task complexity, in order to succeed, we must effectively leverage the pre-training data to learn a general policy that attempts to solve the task, and then specialize it to the target door.

		BC finetuning			Joint training		Target data only		Meta-learning
Task	**PTR (Ours)**	**BC (fine.)**	**Autoreg. BC**	**BeT**	**COG**	**BC**	**CQL**	**BC**	**MACAW**
Take croissant from metal bowl	**7/10**	3/10	5/10	1/10	4/10	4/10	0/10	1/10	0/10
Put sweet potato on plate	**7/20**	1/20	1/20	0/20	0/20	0/20	0/20	0/20	0/20
Place knife in pot	**4/10**	2/10	2/10	0/10	1/10	3/10	3/10	0/10	0/10
Put cucumber in pot	**5/10**	0/10	1/10	0/10	2/10	1/10	0/10	0/10	0/10

Table 24: Performance of PTR and other baseline methods for new tasks in Scenario 3. Note that PTR outperforms all other baselines including BC (finetune), BC with more expressive policy classes (BeT [291], Auto-regressive), offline RL with no pre-training ("Target data only") and joint training [303, 70]. PTR also outperforms few-shot gradient-based meta learning methods such as MACAW [230], which fail to attain non-zero performance.

Interestingly, Table 23 shows that while jointly training on the pre-training and fine-tuning data (or COG [303]) by itself does not outperform BC (joint), the pre-training and fine-tuning approach in PTR leads to significantly better performance, improving over the best BC approach. Since CQL (joint) is equivalent to PTR , but with no Phase 1, this large performance gap indicates the efficacy of offline RL methods trained on large diverse datasets at providing good initializations for learning new downstream tasks. We believe that this finding may be of independent interest to robotic offline RL practitioners: when utilizing multi-task offline RL, it might be better first to run multi-task pre-training followed by fine-tuning, as opposed to jointly training from scratch.

Scenario 3: Learning to solve new tasks in new domains

Finally, we evaluate the efficacy of PTR in learning to solve a new task in a new domain. This scenario presents a generalization requirement that is significantly more challenging than the previously studied scenarios, since both the task and the domain are never seen before. This task is represented via a new task identifier, and pre-training receives no data for this task identifier, or even any data from the kitchen where this task is situated. We pre-train on all 80 pick-and-place style tasks from the bridge dataset, while holding out any data from the new task kitchen, and then fine-tune on 10 demonstrations for 4 target tasks independently in this new kitchen, as shown in Table 24. Methods that utilize more expressive policy architectures (an auto-regressive policy or behavior transformers (BeT) [291]) do not lead to improved performance compared to the standard BC (finetune) approach, and we find that PTR outperforms these approaches. Please find more details on the implementation of auto-regressive BC and BeT in Section G.4.1. This might appear surprising, and perhaps just a hyper-parameter tuning artifact at first, but we present additional qualitative and quantitative analysis aiming at understanding the reasons behind why our offline RL-based PTR approach works better in Section 9.4.4. We also compare to MACAW [230], an offline meta-RL method that utilizes advantage-weighted regression [259] for gradient-based few-shot adaptation, and find that this approach is unable to learn policies that succeed. We discuss the hyperparameter configurations

		Pre-train. rep. + BC finetune	
Task	**PTR (Ours)**	**R3M**	**MAE**
Take croissant from bowl	**7/10**	1/10	3/10
Put sweet potato on plate	**7/20**	0/20	1/20
Place knife in pot	**4/10**	0/10	0/10
Put cucumber in pot	**5/10**	0/10	0/10

Table 25: Performance of PTR and other pre-training methods (R3M and MAE). While both R3M [243] and MAE [346] improve performance over BC on the target data, PTR outperforms both.

that we tried for this approach in Appendix G.3.4. Finally, observe in Table 24 that joint training with CQL or BC, or just using target data, without any pre-training for CQL or BC, all perform significantly worse than PTR.

9.4.3 *Comparison to non-RL Visual Pre-Training Methods*

We also compare PTR to approaches that utilize the diverse bridge dataset or Internet-scale data for task-agnostic visual representation learning, followed by down-stream behavioral cloning only on the target fine-tuning task which utilizes the representation learned during pre-training. In particular, we compare to two approaches: R3M [243], which utilizes the Ego4D dataset of human videos to obtain a representation, and MVP [271, 346], which trains a masked auto-encoder [130] on the Bridge Dataset and utilizes the learned latent space as the representation of the new image. Observe in Table 25 that, while utilizing R3M or MAE does improve over running BC on the target data alone (compare R3M and MAE in Table 25 to BC on target data only in Table 24), the pre-training scheme from PTR outperforms both of these prior pre-training approaches, indicating the efficacy of offline RL pre-training on diverse robot data in recovering useful representations for downstream policy learning.

9.4.4 *Understanding the Benefits of PTR over BC*

One natural question to ask given the results in this paper is: why does utilizing an offline RL method for pre-training and fine-tuning as in PTR outperform BC-based methods even though the dataset is quite "BC-friendly", consisting of only demonstrations? The answer to this question is not obvious, especially since joint training with BC still outperforms jointly training with CQL on both pre-training and target demonstration data (COG) in our results in Table 24.

To understand the reason behind improvements from RL, we perform a qualitative evaluation of the policies learned by PTR and BC (finetune) on two tasks: take croissant from metal bowl and put cucumber in bowl in Figure 6. We find that the failure mode of BC policies can be primarily explained as a lack of precision in locating the object, or

a prematurely-executed grasping action. This is especially prevalent in settings where the object of interest is farther away from the robot gripper at the initial state, and hints at the inability of BC to prioritize learning the critical decisions (e.g., precisely moving over the object before the grasping action) over non-critical ones (e.g., the action to take to reach nearby the object from farther away). On the other hand, RL can learn to make such critical decisions correctly as shown in Figure 6. We present additional rollouts in Appendix G.2.

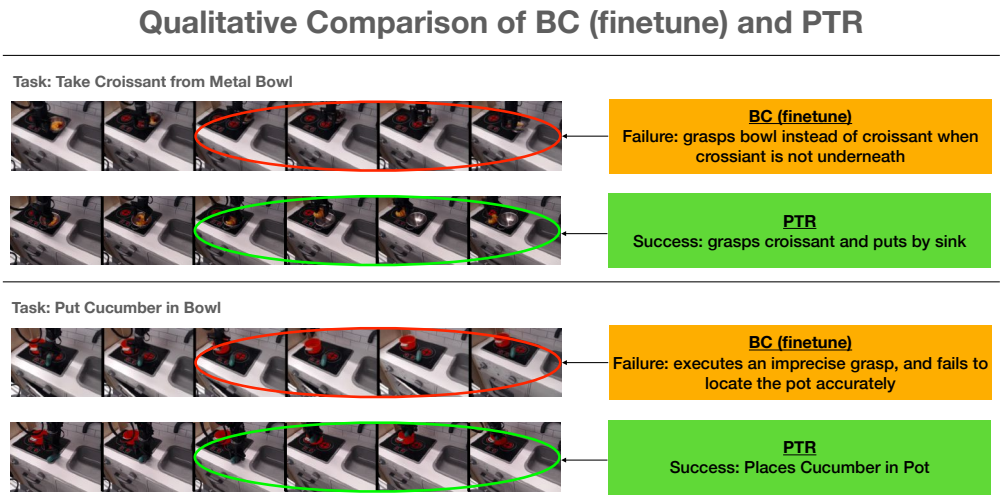


Figure 6: Qualitative successes of PTR visualized alongside failures of BC (fine-tune). As an example, observe that while PTR is accurately able to reach to the croissant and grasp it to solve the task, BC (finetune) is imprecise and grasps the bowl instead of the croissant resulting in failure.

Task	BC (finetune)	PTR	AW-BC (finetune)
Cucumber	0/10	5/10	5/10
Croissant	3/10	7/10	6/10

Table 26: Performance of advantage-weighted BC on tasks from Table 24. Observe that weighting BC using advantage estimates from the Q-function learned by PTR leads to much better performance than standard BC (finetune), almost recovering PTR performance. This test indicates that the Q-function in PTR allows us to be accurate on the more critical decisions, thereby preventing the failures of BC.

Next, to verify if the performance benefits can be explained by the ability of Q-learning to prioritize critical decisions, we run a form of weighted behavioral cloning, where the weights $w_\phi(\boldsymbol{s}, \boldsymbol{a})$ are derived from the *advantage estimates computed using a frozen Q-function learned by PTR* after fine-tuning:

$$w_\phi(\boldsymbol{s}, \boldsymbol{a}) \propto \exp(Q_\phi(\boldsymbol{s}, \boldsymbol{a}) - \max_{\boldsymbol{a}'} Q_\phi(\boldsymbol{s}, \boldsymbol{a}')).$$

Note that this is not the same as standard advantage-weighted regression [259], which uses Monte-Carlo return estimates for computing advantage weights instead of using

advantages computed under a Q-function trained via PTR or CQL. As shown in Table 26, we find that this advantage-weighted BC (AW-BC) approach performs significantly better than BC (finetune) method and comparably to PTR, for two tasks (croissant and cucumber from Table 24). Since AW-BC is essentially the same as BC, just with a modified weight to indicate the importance of any transition, this performance improvement clearly indicates the benefits of learning value functions via PTR in a pre-training then fine-tuning setting, even when we only have demonstration data. Note that since AW-BC uses the PTR-derived weights after fine-tuning, it cannot serve as an independent method, but rather amounts to another way to use the PTR value function.

9.4.5 *Effective Use of High-Capacity Neural Networks*

To understand the importance of designing techniques that enable us to use high-capacity models for offline RL, we examine the efficacy of PTR with different neural network architectures on the open door task from Scenario 2, and the put cucumber in pot and take croissant out of metallic bowl tasks from Scenario 3. We compare to standard three-layer convolutional network architectures used by prior work for Deepmind control suite tasks (see for example, Kostrikov et al. [174]), an IMPALA [74] ResNet that consists of 15 convolutional layers spread across a stack of 3 residual blocks, and the ResNet 18, 34, and 50 architectures with our proposed design decisions. Observe in Figure 7 that the performance of smaller networks (Small, IMPALA) is significantly worse than the ResNet in the door opening task. For the pick-and-place tasks that contain a much larger dataset, IMPALA and ResNet18 all perform much worse than ResNet 34 and 50.

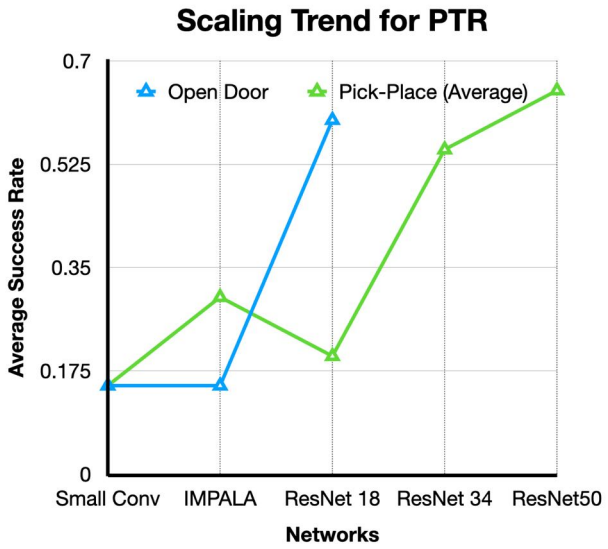


Figure 7: Scaling trends for PTR on the open door task, and average over two pick & place tasks from Scenario 3. Note that with our design decisions, PTR is able to effectively benefit from high capacity networks.

9.4.6 *Autonomous Online Fine-Tuning*

So far, we've evaluated PTR with offline fine-tuning to new tasks. However, by pre-training representations with offline RL, we can also enable autonomous improvement through online RL fine-tuning. In this section, we will demonstrate this benefit by showing that an offline initialization learned by PTR pre-training can be effectively fine-tuned autonomously with online rollouts. This procedure provides a way forward to build

	SACfD	PTR (offline → online)
All positions	0% → 0%	**53% → 73%**
Novel OOD positions	0% → 0%	**13% → 60%**

Table 27: Performance before and after online fine-tuning. The success rate of the PTR pre-trained policy is improved significantly from online fine-tuning, especially on novel out-of-distribution (OOD) initial positions that must be learned entirely from autonomous interaction in the real world. The results are reported as the average of 3 trials from each initial position.

self-improving robotic RL systems that bring the best of diverse robotic datasets and learning via online interaction.

Task. For this experiment, we consider the "open door" task from Scenario 2. Our goal is to improve the success rate of the learned policy obtained after PTR pre-training and offline fine-tuning using autonomous online rollouts from ten initial positions. These ten initial positions consist of five positions obtained by randomly sampling from the target demonstrations used for offline fine-tuning, and five more challenging **out-of-distribution initial positions**, that were never seen before.

Reward functions. To run RL, we need a mechanism to annotate online rollouts with rewards. Following prior works [301, 162], we trained a neural-network binary classifier to detect a given visual observation as a success (+1 reward) or failure (0 reward) and use it to annotate rollouts executed during online interaction.

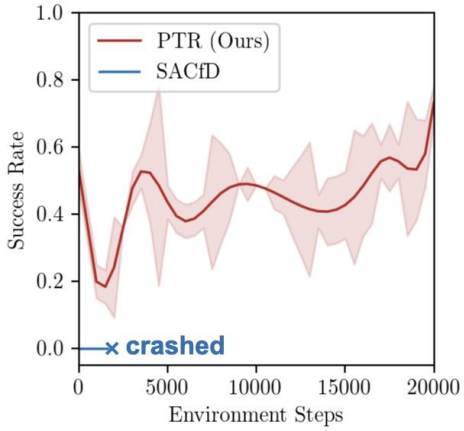


Figure 8: Online fine-tuning for PTR on the open door task. PTR improves the success rate of the pre-trained policy from 53% to 73% (from 13% to 60% for the harder positions), while SACfD crashes due to unsafe behavior during exploration. We ran PTR online fine-tuning for 2 seeds in the real world.

Reset policy. To run online fine-tuning autonomously without any human intervention in the real world, we also need a "reset policy" that closes the door after a successful online rollout. To this end, we also pre-trained a close-door policy separately, which is used only for resetting the door. Note that online fine-tuning only fine-tunes the open-door policy, while the reset policy is kept fixed throughout.

Online training setup. Equipped with the reset policy and the reward classifier, we are able to run online fine-tuning in the real world. Starting from the pre-trained policy obtained via PTR, our method alternates between collecting a new trajectory and taking gradient steps. The update-to-data ratio [46] is set to 10, which means that we make 10

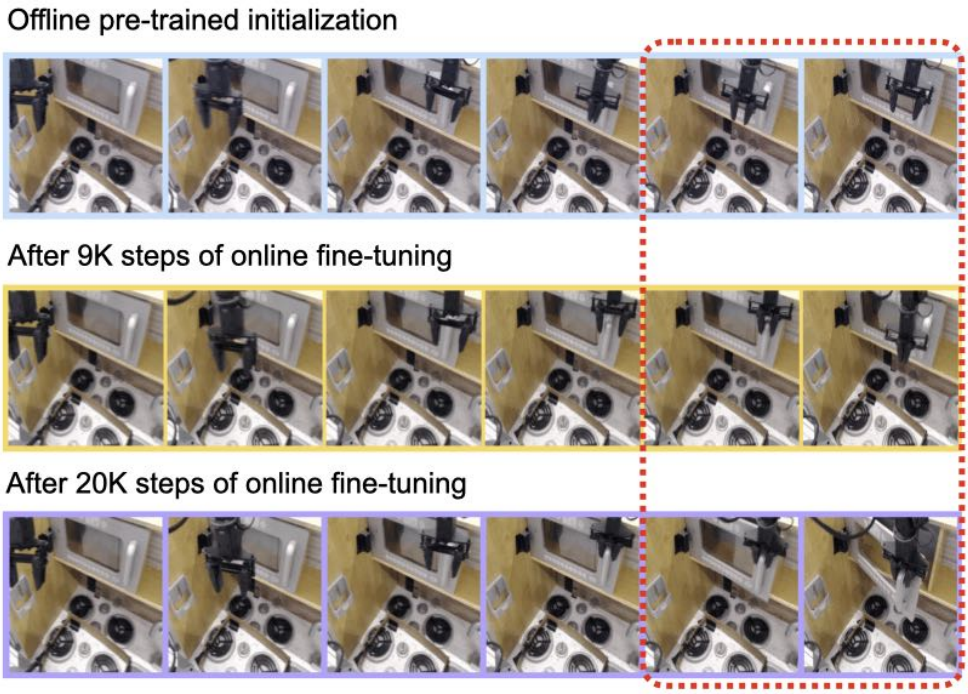


Figure 9: Evolution of learned behaviors during autonomous online fine-tuning of PTR starting from one of the hard initial positions. The blue box illustrates that the offline initialization fails to grasp the handle. After 9K steps of online interaction, it successfully grasps the handle but fails to open the door. After 20K steps, it learns to successfully open the door.

gradient updates for every environment step. More details about our implementation and evaluations can be found in Appendix G.6.

Results. We compare our method with a prior method that trains SAC [126] from scratch using both online data and offline demonstrations (denoted by "SACfD"). This approach is an improved version of DDPGfD [331] which uses a stronger off-policy RL algorithm (SAC). We present the learning curve during the online fine-tuning in Figure 8, and the success rates before and after fine-tuning in Table 27. As shown in Figure 8, it was difficult to run SACfD over a long time on the robot, as the system crashes due to unsafe actions during exploration (pictures shown in Appendix G.6). In contrast, the pre-trained PTR policy is able to perform online exploration in a stable manner, and improve the success rate of the pre-trained policy within 20K steps of online interaction. Specifically, this boost in performance stems from learning to solve the task from **3/5** of the more challenging, out-of-distribution initial positions, that were never seen before in the prior data, as shown in Figure 9.

9.5 Related Work

Going beyond methods that only perform fine-tuning from a learned initialization with online interaction [241, 175, 200] as we studied in the previous section of this dissertation,

we consider two independent fine-tuning settings in this chapter: (1) the setting where we do not use any online interaction and fine-tune the pre-trained policy entirely offline, (2) the setting where a limited amount of online interaction is allowed to autonomously acquire the skills to solve the task from a challenging initial condition. This resembles the problem setting considered by offline meta-RL methods [207, 63, 230, 266, 213]. However, our approach is simpler as we fine-tune the very same offline RL algorithm that we use for pre-training, without any distinct meta updates. In our experiments, we observe that our method, PTR, outperforms the meta-RL method of Mitchell et al. [230].

Some other prior approaches that attempt to leverage large, diverse datasets via representation learning [226, 354, 355, 243, 130, 346, 222], as well as other methods for learning from human demonstrations, such as behavioral cloning methods with expressive policy architectures [291]. We compare to some of these methods [346, 243] in our experiments and find that PTR outperforms these methods. We also perform an empirical study to identify the design decisions behind the improved performance of RL-based PTR on demonstration data compared to BC, and find that the gains largely come from the ability of the value function in identifying the most "critical" decisions in a trajectory.

The most closely related to our approach are prior methods that run model-free offline RL on diverse real-world data and then fine-tune on new tasks [303, 162, 154, 41, 197]. These prior methods typically only consider the setting of *online* fine-tuning, whereas in our experiments, we demonstrate the efficacy of PTR for offline fine-tuning (where we must acquire a good policy for the downstream task using 10-15 demonstrations) *as well as* online fine-tuning considered in these prior works, where we must acquire a new task entirely via autonomous interaction in the real world.

9.6 Discussion and Limitations

We presented a system that uses diverse prior data for general-purpose offline RL pre-training, followed by fine-tuning to downstream tasks. The prior data, sourced from a publicly available dataset, consists of over a hundred tasks across ten scenes and our policies can be fine-tuned with as few as 10 demonstrations. We show that this approach outperforms prior pre-training and fine-tuning methods based on imitation learning. One of the most exciting directions for future work is to further scale up this pre-training to provide a single policy initialization, that can be utilized as a starting point, similar to GPT3 [34]. An exciting future direction is to scale PTR up to more complex settings, including to novel robots and incorporate training from diverse sources of robot video data. Since joint training with offline RL was worse than pre-training and then fine-tuning with PTR, another exciting direction for future work is to understand the pros and cons of joint training and fine-tuning in the context of robot learning.

10

HARDWARE ACCELERATOR DESIGN

Abstract

In this chapter, we utilize offline RL to the problem of designing hardware accelerators, i.e., chips that are tailored towards improving the efficiency of running certain software applications. For instance, TPU accelerators aim to optimize the performance of machine learning models. While a paradigm shift towards specializing hardware is already starting to show promising results, designers still need to spend considerable manual effort and perform large number of time-consuming simulations to find accelerators that can accelerate multiple target applications while obeying design constraints. Moreover, a designer would need to typically start from scratch every time the set of target applications or design constraints change in such a "simulation-driven" workflow. An alternative paradigm is to use an offline approach that utilizes logged simulation data, to architect hardware accelerators, without needing active simulations. This is different from supervised learning: instead of predicting a performance metric associated with an accelerator, we wish to find a new accelerator that attains near-optimal metrics, much in the same way where we wish to find a near-optimal policy in offline RL. In this problem, our offline RL based approach, dubbed PRIME , not only alleviates the need to run time-consuming simulation, but also enables data reuse and applies even when set of target applications changes. PRIME architects accelerators for both single and multiple applications, improving performance upon state-of-the-art simulation-driven methods by about 1.54× and 1.20×, while reducing the total simulation time by 93% and 99%, respectively.

10.1 INTRODUCTION

The death of Moore's Law [73] has driven the growth of specialized hardware accelerators. These specialized accelerators are tailored to specific applications [360, 275, 246, 295]. To design specialized accelerators, akin to standard online RL, designers first spend considerable amounts of time developing simulators that closely model the real accelerator

performance, and then optimize the accelerator using the simulator. While such simulators can automate accelerator design, this requires a large number of simulator queries for each new design, both in terms of simulation time and compute requirements, and this cost increases with the size of the design space [361, 295, 135]. Moreover, most of the accelerators in the design space are typically infeasible [135, 360] because of build errors in silicon or compilation/mapping failures. When the target applications change or a new application is added, the complete simulation-driven procedure is generally repeated. To make such approaches efficient and practically viable, designers typically "bake-in" constraints or otherwise narrow the search space, but such constraints can leave out high-performing solutions [54, 255, 39].

An alternate approach, proposed in this chapter, is to devise an *offline* optimization method, based on offline RL techniques, that only utilizes a database of previously tested accelerator designs, annotated with measured performance metrics, to produce new optimized designs *without* additional active queries to an explicit silicon or a cycle-accurate simulator. In the context of hardware accelerator design, such an offline approach provides three key benefits: **(1)** it significantly shortens the recurring cost of running large-scale simulation sweeps, **(2)** it alleviates the need to explicitly bake in domain knowledge or search space pruning, and **(3)** it enables data re-use by empowering the designer to optimize accelerators for new unseen applications, by the virtue of effective generalization. While data-driven approaches have shown promising results in biology [87, 33, 326], using offline optimization methods to design accelerators has been challenging primarily due to the abundance of infeasible design points [360, 135].

The key contribution of this chapter is PRIME , a conservative value-based approach to automatically architect high-performing application-specific accelerators by using only previously collected static data. Since the problem of accelerator design requires us to only select one action, and not a sequence of actions, we do not need to learn a complete value function. Instead, it suffices for PRIME to learn a conservative model of the one-step reward or objective function from the offline dataset (e.g., latency or power consumption of an accelerator). Then, PRIME finds high-performing application-specific accelerators by optimizing the architectural parameters against this learned conservative model, as shown in Figure 10. Akin to the full sequential setting, while optimizing naïvely trained models of the objective function usually produces poor-performing, out-of-distribution designs that erroneously appear optimistic [178, 33, 326], the conservative model in PRIME gives rise to good designs. Furthermore, in contrast to prior works that discard infeasible points [135, 326], our proposed method instead incorporates infeasible points when learning the conservative surrogate model by treating them as additional negative samples. During evaluation, PRIME optimizes the learned conservative model.

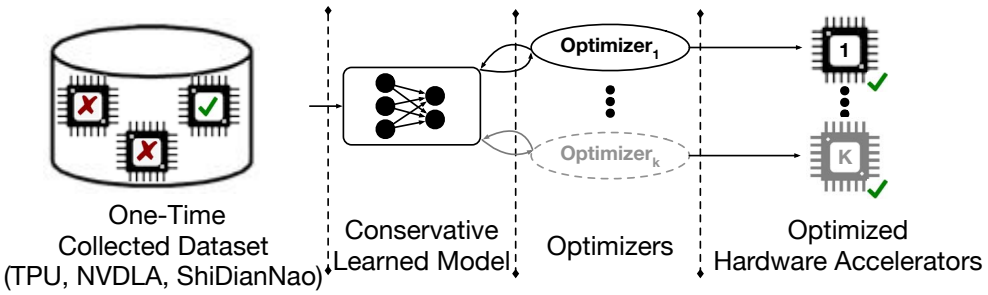


Figure 10: Overview of PRIME . We use a one-time collected dataset of prior accelerator designs, including TPU-style [361], NVDLA-style [249], and ShiDianNao-style [67] accelerators to train a conservative surrogate model, which is used to design accelerators to meet desired goals and constraints.

Our results show that PRIME architects hardware accelerators that improve over the best design in the training dataset, on average, by 2.46× (up to 6.7×) when specializing for a single application. In this case, PRIME also improves over the best conventional simulator-driven optimization methods by 1.54× (up to 6.6×). These performance improvements are obtained while reducing the total simulation time to merely 7% and 1% of that of the simulator-driven methods for single-task and multi-task optimization, respectively. More importantly, a contextual version of PRIME can design accelerators that are *jointly optimal* for a set of *nine* applications without requiring any additional domain information. In this challenging setting, PRIME improves over simulator-driven methods, which tend to scale poorly as more applications are added, by 1.38×. Finally, we show that the models trained with PRIME on a set of training applications can be readily used to obtain accelerators for *unseen* target applications, without any retraining on the new application. Even in this *zero-shot* optimization scenario, PRIME outperforms simulator-based methods that require re-training and active simulation queries by up to 1.67×. In summary, PRIME allows us to effectively address the shortcomings of simulation-driven approaches, it: (1) significantly reduces the simulation time, (2) enables data reuse and enjoys generalization properties, and (3) does not require domain-specific engineering or search space pruning.

10.2 Background on Hardware Accelerators

The goal of specialized hardware accelerators—Google TPUs [153, 121], Nvidia GPUs [250], GraphCore [111]—is to improve the performance of specific applications, such as machine learning models. To design such accelerators, architects typically create a parameterized design and sweep over parameters using simulation.

Target hardware accelerators. Our primary evaluation uses an industry-grade and highly parameterized template-based accelerator following prior work [361]. This template enables architects to determine the organization of various components, such as compute units, memory cells, memory, etc., by searching for these configurations in a discrete

design space. Some ML applications may have large memory requirements (e.g., large language models [34]) demanding sufficient on-chip memory resources, while others may benefit from more compute blocks. The hardware design workflow directly selects the values of these parameters. In addition to this accelerator and to further show the generality of our method to other accelerator design problems, we evaluate two distinct dataflow accelerators with different search spaces, namely NVDLA-style [249] and ShiDianNao-style [67] from Kao et al. [164] (See Section 10.6 and Appendix H.3 for a detailed discussion; See Table 33 for results).

How does an accelerator work? We briefly explain the computation flow on our template-based accelerators (Figure 11) and refer the readers to Appendix H.3 for details on other accelerators. This template-based accelerator is a 2D array of processing elements (PEs). Each PE is capable of performing matrix multiplications in a single instruction multiple data (SIMD) paradigm [110]. A controller orchestrates the data transfer (both activations and model parameters) between off-chip DRAM memory and the on-chip buffers and also reads in and manages the instructions (e.g. convolution, pooling, etc.) for execution. The computation stages on such accelerators start by sending a set of activations to the compute lanes, executing them in SIMD manner, and either storing the partial computation results or offloading them back into off-chip memory. Compared to prior works [135, 67, 164], this parameterization is unique—it includes multiple compute lanes per each PE and enables SIMD execution model within each compute lane—and yields a distinct accelerator search space accompanied by an end-to-end simulation framework.

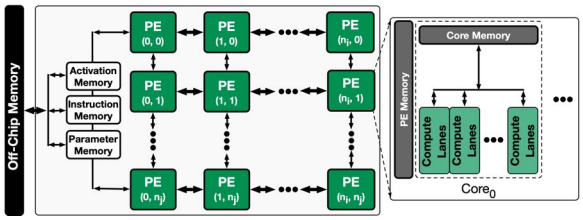


Figure 11: An industry-level machine learning accelerator [361].

10.3 Problem Statement, Training Data and Evaluation Protocol

Our template-based parameterization maps the accelerator, denoted as $\boldsymbol{x}$, to a discrete design space, $\boldsymbol{x} = [x_1, x_2, \cdots, x_K]$, and each x_i is a discrete-valued variable representing one component of the microarchitectural template, as shown in Table 28 (See Appendix H.3 for the description of other accelerator search spaces studied in our work). In our context, the accelerator $\boldsymbol{x}$ plays the same role as an action $\boldsymbol{a}$.

An accelerator design maybe be infeasible due to various reasons, such as a compilation failure or the limitations of physical implementation, and we denote the set of all such feasibility criterion as Feasible($\boldsymbol{x}$). The feasibility criterion depends on both the target software and the underlying hardware, and it is not easy to identify if a given $\boldsymbol{x}$ is infeasible without running explicit simulation. We will require our design procedure to

Table 28: The accelerator design space parameters for the primary accelerator search space targeted in this work. The maximum possible number of accelerator designs (including feasible and infeasible designs) is 452,760,000. Cal-QL only uses a small randomly sampled subset of the search space.

Accelerator Parameter	# Discrete Values	Accelerator Parameter	# Discrete Values
# of PEs-X	10	# of PEs-Y	10
PE Memory	7	# of Cores	7
Core Memory	11	# of Compute Lanes	10
Instruction Memory	4	Parameter Memory	5
Activation Memory	7	DRAM Bandwidth	6

not only learn the value of the objective function but also to learn to navigate through infeasible solutions to performant feasible solutions $\boldsymbol{x}^*$ satisfying $\text{Feasible}(\boldsymbol{x}^*) = 1$.

Our training dataset $\mathcal{D}$ consists of a modest set of accelerators $\boldsymbol{x}_i$ that are randomly sampled from the design space and evaluated by the hardware simulator. We partition the dataset $\mathcal{D}$ into two subsets, $\mathcal{D}_{\text{feasible}}$ and $\mathcal{D}_{\text{infeasible}}$. Let $f(\boldsymbol{x})$ denote the desired objective (e.g., latency, power, etc.) we intend to optimize over the space of accelerators $\boldsymbol{x}$. We do not possess functional access to $f(\boldsymbol{x})$, and the optimizer can only access $f(\boldsymbol{x})$ values for accelerators $\boldsymbol{x}$ in the feasible partition of the data, $\mathcal{D}_{\text{feasible}}$. For all infeasible accelerators, the simulator does not provide any value of $f(\boldsymbol{x})$. In addition to satisfying feasibility, the optimizer must handle explicit constraints on parameters such as area and power [85]. In our applications, we impose an explicit area constraint, $\text{Area}(\boldsymbol{x}) \leq a_0$, though additional explicit constraints are also possible. To account for different constraints, we formulate this task as a constrained optimization problem. Formally:

$$\begin{aligned} &\min_{\boldsymbol{x}} \; f(\boldsymbol{x}) \quad \text{s.t.} \quad \text{Area}(\boldsymbol{x}) \leq a_0, \quad \text{Feasible}(\boldsymbol{x}) = 1 \\ &\quad \text{on} \;\; \mathcal{D} = \mathcal{D}_{\text{feasible}} \cup \mathcal{D}_{\text{infeasible}} = \{(\boldsymbol{x}_1, \boldsymbol{y}_1), \cdots, (\boldsymbol{x}_N, \boldsymbol{y}_N)\} \cup \{\boldsymbol{x}'_1, \cdots, \boldsymbol{x}'_{N'}\} \end{aligned} \tag{10.3.1}$$

While Equation 10.3.1 may appear similar to a typical black-box optimization problem, solving it over the space of accelerator designs is challenging due to the large number of infeasible points, the need to handle explicit design constraints, and the difficulty in navigating the non-smooth landscape (See Figure 12 and Figure 56 in the Appendix) of the objective function.

What makes optimization over accelerators challenging? Compared to other domains where model-based optimization methods have been applied [33, 326], optimizing accelerators introduces a number of practical challenges. First, accelerator design spaces typically feature a narrow manifold of feasible accelerators within a sea of infeasible points [246, 295, 103], as visualized in Figure 12 and Appendix (Figure 57). While some of these infeasible points can be identified via simple rules (e.g. estimating chip area usage), most infeasible points correspond to failures during compilation or hardware

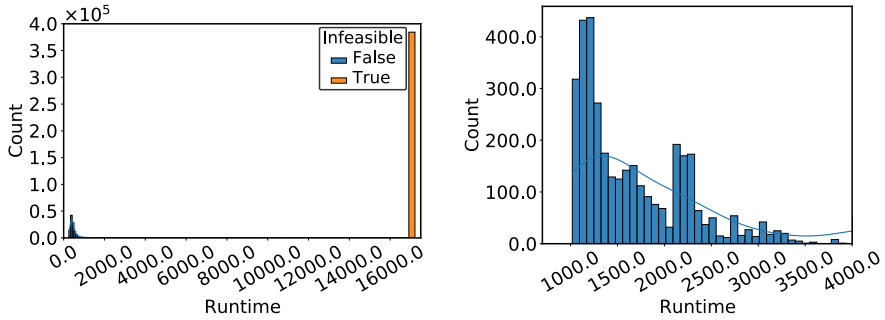


Figure 12: Left: histogram of infeasible (right orange bar with large score values) and feasible (left cluster of bars) data points for MobileNetEdgeTPU; **Right:** zoomed-in histogram (different number of bins) focused on feasible points highlighting the variable latencies.

simulation. These infeasible points are generally not straightforward to formulate into the optimization problem and requires simulation [295, 255, 360].

Second, the optimization objective can exhibit high sensitivity to small variations in some architecture parameters (Figure 56b) in some regions of the design space, but remain relatively insensitive in other parts, resulting in a complex optimization landscape. This suggests that optimization algorithms based on local parameter updates (e.g., gradient ascent, evolutionary schemes, etc.) may have a challenging task traversing the nearly flat landscape of the objective, which can lead to poor performance.

Training dataset. We used an offline dataset $\mathcal{D}$ of (accelerator parameters, latency) via random sampling from the space of 452M possible accelerator configurations. Our method is only provided with a relatively modest set of feasible points ($\leq$ 8000 points) for training, and these points are the *worst-performing* feasible points across the pool of randomly sampled data. This dataset is meant to reflect an easily obtainable and an application-agnostic dataset of accelerators that could have been generated once and stored to disk, or might come from real physical experiments. We emphasize that no assumptions or domain knowledge about the application use case was made during dataset collection. Table 29 depicts the list of target applications, evaluated in this work, includes three variations of MobileNet [121, 282, 141], three in-house industry-level models for object detection (M4, M5, M6; names redacted to prevent anonymity violation), a U-net model [278], and two RNN-based encoder-decoder language models [113, 133, 281, 206]. These applications span the gamut from small models, such as M6, with only 0.4 MB model parameters that demands less on-chip memory, to the medium-sized models ($\geq$ 5 MB), such as MobileNetV3 and M4 models, and large models ($\geq$ 19 MB), such as t-RNNs, hence requiring larger on-chip memory.

Evaluation protocol. To compare simulator-driven methods and our data-driven method, we limit the number of feasible points (costly to evaluate) that can be used by any algorithm to equal amounts. We still provide infeasible points to any method and leave it

Table 29: Description of applications, their domains, number of (convolutions, depth-wise convolutions, feed-forward) XLA ops, model parameter size, instruction sizes in bytes, number of compute operations.

Name	Domain	# of XLA Ops (Conv, D/W, FF)	Model Param	Instr. Size	# of Compute Ops.
MobileNetEdgeTPU	Image Class.	(45, 13, 1)	3.87 MB	476,736	1,989,811,168
MobileNetV2	Image Class.	(35, 17, 1)	3.31 MB	416,032	609,353,376
MobileNetV3	Image Class.	(32, 15, 17)	5.20 MB	1,331,360	449,219,600
M4	Object Det.	(32, 13, 2)	6.23 MB	317,600	3,471,920,128
M5	Object Det.	(47, 27, 0)	2.16 MB	328,672	939,752,960
M6	Object Det.	(53, 33, 2)	0.41 MB	369,952	228,146,848
U-Net	Image Seg.	(35, 0, 0)	3.69 MB	224,992	13,707,214,848
t-RNN Dec	Speech Rec.	(0, 0, 19)	19 MB	915,008	40,116,224
t-RNN Enc	Speech Rec.	(0, 0, 18)	21.62 MB	909,696	45,621,248

up to the optimization method to use it or not. This ensures our comparisons are fair in terms of the amount of data available to each method. However, it is worthwhile to note that in contrast to our method where *worse-quality* data points from small offline dataset are used, the simulator-driven methods have an inherent advantage because they can steer the query process towards the points that are more likely to be better in terms of performance. Following prior work in data-driven design [33], we evaluate each run of a method by first sampling the top $n = 256$ design candidates according to the algorithm's predictions, evaluating all of these under the ground truth objective function and recording the performance of the best accelerator design. The final reported results is the median of ground truth objective values across five independent runs.

10.4 PRIME : ARCHITECTING ACCELERATORS VIA CONSERVATIVE MODELS

As shown in Figure 13, our method first learns a conservative surrogate model of the optimization objective using the offline dataset. Then, it optimizes this learned model using a discrete optimizer. The optimization process does not require access to a simulator, nor to real-world experiments beyond the initial dataset, except when evaluating the final top-performing $n = 256$ designs (Section 10.3). This is built on the principle of conservative value estimation for offline RL from Chapter 5: while conservative value estimation methods that we have discussed so far in this dissertation attempt to estimate a pessimistic value-function, our approach PRIME is a special sub-case that involves a single decision-making step (as opposed to making a sequence of decisions).

10.4.1 *Learning Conservative Models Using Logged Offline Data*

Our goal is to utilize a logged dataset of feasible accelerator designs labeled with the desired performance metric (e.g., latency), $\mathcal{D}_{\text{feasible}}$, and additional infeasible designs, $\mathcal{D}_{\text{infeasible}}$ to learn a mapping $f_\theta : \mathcal{X} \to \mathbb{R}$, that maps the accelerator configuration x to its corresponding metric y. This learned surrogate model can then be optimized by the optimizer. While a straightforward approach for learning such a mapping is to train it

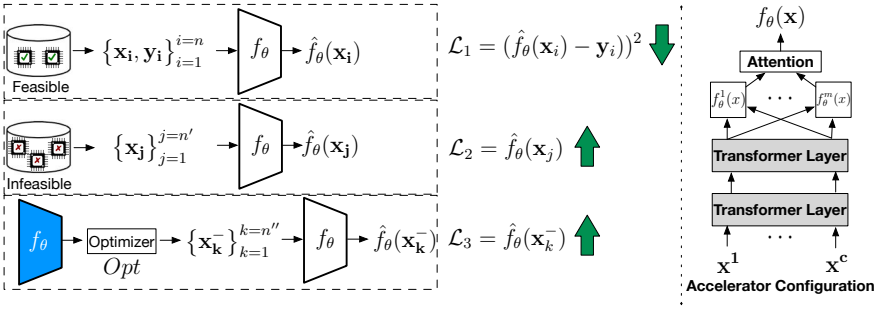


Figure 13: Overview of PRIME which trains a conservative model $f_\theta(\boldsymbol{x}_i)$ using Equation 10.4.2. Our neural net model for $f_\theta(\boldsymbol{x})$ utilizes two transformer layers [330], and a multi-headed architecture which is pooled via a soft-attention layer.

via supervised regression (which is equivalent to standard temporal-difference learning for value estimation in one-step decision-making problems, where an episode terminates as soon as one action is performed and a terminal reward is collected), by minimizing the mean-squared error $\mathbb{E}_{\boldsymbol{x}_i, y_i \sim \mathcal{D}}[(f_\theta(\boldsymbol{x}_i) - y_i)^2]$, as we have seen in this dissertation and in other prior works [178], such predictive models can arbitrarily overestimate the value of an unseen input $\boldsymbol{x}_i$. This can cause the optimizer to find a solution $\boldsymbol{x}^*$ that performs poorly in the simulator but looks promising under the learned model. We empirically validate this overestimation hypothesis and find it to confound the optimizer in on our problem domain as well (See Figure 58 in Appendix).

To prevent overestimated values at unseen inputs from confounding the optimizer, we follow the conservative value estimation recipe from Chapter 5 (specifically, Equation 5.1.2) and train $f_\theta(\boldsymbol{x})$ with an additional term that explicitly maximizes the function value $f_\theta(\boldsymbol{x})$ at out-of-distribution $\boldsymbol{x}$ values. Note that instead of minimizing the value $f_\theta(\boldsymbol{x})$ on unseen $\boldsymbol{x}$, we maximize this value because the problem in Equation 10.3.1 requires us to find the minimum of the ground-truth function. Such unseen designs $\boldsymbol{x}$ are "negative mined" by running a few iterations of a stochastic optimization procedure that aims to maximize f_θ in the inner loop. In the context of this single-step decision-making problem, this procedure is analogous to adversarial training [109]. Equation 10.4.1 formalizes this objective:

$$\theta^* := \arg\min_\theta \quad \mathcal{L}(\theta) := \mathbb{E}_{\boldsymbol{x}_i, y_i \sim \mathcal{D}_{\text{feasible}}} \left[\left(f_\theta(\boldsymbol{x}_i) - y_i \right)^2 \right] - \alpha \mathbb{E}_{\boldsymbol{x}_i^- \sim \text{Opt}(f_\theta)} \left[f_\theta(\boldsymbol{x}_i^-) \right]. \quad (10.4.1)$$

$\boldsymbol{x}_i^-$ denotes the negative samples produced from an optimizer $\text{Opt}(\cdot)$ that attempts to maximize the current learned objective model, f_θ. We will discuss our choice of Opt in the Appendix H.2.

10.4.2 *Incorporating Design Constraints by Training on Infeasible Points*

While conservative value estimation methods provide us with a recipe for optimizing Equation 10.4.1 via a conservative surrogate model, this is not enough when optimizing

over accelerators, as we will also show empirically (Appendix H.1.1). This is because while explicitly minimizing for out-of-distribution designs constrains the design procedure to the data, it does not provide any information about accelerator design constraints. Fortunately, this information can be provided by infeasible points, $\mathcal{D}_{\text{infeasible}}$. The training procedure in Equation 10.4.1 provides a simple way to do incorporate such infeasible points: we simply incorporate $\boldsymbol{x}_i' \sim \mathcal{D}_{\text{infeasible}}$ as additional out-of-distribution samples and maximize the prediction at these points. This gives rise to our final objective:

$$\min_{\theta} \quad \mathcal{L}^{\text{inf}}(\theta) := \mathcal{L}(\theta) - \beta \mathbb{E}_{\boldsymbol{x}_i' \sim \mathcal{D}_{\text{infeasible}}} \left[f_\theta(\boldsymbol{x}_i') \right] \tag{10.4.2}$$

10.4.3 *Optimizing Multiple Applications and Zero-Shot Design*

One of the central benefits of an offline learning approach is that it enables learning powerful models that generalize over the space of applications, potentially being effective for new unseen application domains. In our experiments, we evaluate PRIME on designing accelerators for multiple applications denoted as $k = 1, \cdots, K$, jointly or for a novel unseen application. In this case, we utilized a dataset $\mathcal{D} = \{\mathcal{D}_1, \cdots, \mathcal{D}_K\}$, where each $\mathcal{D}_k$ consists of a set of accelerator designs, annotated with the latency value and the feasibility criterion for a given application k. While there are a few overlapping designs in different parts of the dataset annotated for different applications, most of the designs only appear in one part. To train a single conservative model $f_\theta(\cdot)$ for multiple applications, we extend the training procedure in Equation 10.4.2 to incorporate *context vectors* $\boldsymbol{c}_k \in \mathbb{R}^d$ for various applications driven by a list of application properties in Table 29. A context vector is akin to a state in the context of full sequential reinforcement learning.

The learned function in this setting is now conditioned on the context $f_\theta(\boldsymbol{x}, \boldsymbol{c}_k)$. We train f_θ via the objective in Equation 10.4.2, but in expectation over all the contexts and their corresponding datasets: $\min_\theta \mathbb{E}_{k \sim [K]} \left[\mathcal{L}_k^{\text{inf}}(\theta) \right]$. Once such a contextual model is learned, we can either optimize the average models across a set of contexts $\{\boldsymbol{c}_i, \boldsymbol{c}_2, \cdots, \boldsymbol{c}_n\}$ to obtain an accelerator that is optimal for multiple applications simultaneously on an average ("multi-model" optimization), or optimize this contextual model for a novel context vector, corresponding to an unseen application ("zero-shot" generalization). In this case, PRIME is not allowed to train on any data corresponding to this new unseen application. While such zero-shot generalization might appear surprising at first, note that the context vectors are not simply one-hot vectors, but consist of parameters with semantic information, which the conservative model can generalize over.

Learned conservative model optimization. Prior work [360] has shown that the most effective optimizers for accelerator design are meta-heuristic/evolutionary optimizers. We therefore choose to utilize, firefly [358, 359, 215] to optimize our conservative model. This algorithm maintains a set of optimization candidates (a.k.a. "fireflies") and jointly update

them towards regions of low objective value, while adjusting their relative distances appropriately to ensure multiple high-performing, but diverse solutions. We discuss additional details in Appendix H.2.1.

Cross validation: which model and checkpoint should we evaluate? Similarly to supervised learning, models trained via Equation 10.4.2 can overfit, leading to poor solutions. Thus, we require a procedure to select which hyperparameters and checkpoints should actually be used for the design. This is crucial, because we cannot arbitrarily evaluate as many models as we want against the simulator. While effective methods for model selection have been hard to develop in offline reinforcement learning [326, 327], we devised a simple scheme using a validation set for choosing the values of α and β (Equation 10.4.2), as well as which checkpoint to utilize for generating the design. For each training run, we hold out the best 20% of the points out of the training set and use them *only* for cross-validation as follows. Typical cross-validation strategies in supervised learning involve tracking validation error (or risk), but since our model is trained conservatively, its predictions may not match the ground truth, making such validation risk values unsuitable for our use case. Instead, we track Kendall's ranking correlation between the predictions of the learned model $f_\theta(\boldsymbol{x}_i)$ and the ground truth values y_i (Appendix H.2) for the held-out points for each run and then pick values of α, β and the checkpoint that attain the highest validation ranking correlation. We present the pseudo-code for PRIME (Algorithm 9) and implementation details in Appendix H.2.1.

10.5 Related Work

Optimizing hardware accelerators has become more important recently. Prior works [258, 143, 311, 246, 50, 170, 19, 13, 135, 332, 352] mainly rely on expensive-to-query hardware simulators to navigate the search space and/or target single-application accelerators. For example, HyperMapper [246] targets compiler optimization for FPGAs by continuously interacting with the simulator in a design space with relatively few infeasible points. Mind Mappings [135], optimizes software mappings to a fixed hardware provided access to millions of feasible points and throws away infeasible points during learning. MAGNet [332] uses a combination of pruning heuristics and online Bayesian optimization to generate accelerators for image classification models in a single-application setting. AutoDNNChip [352] uses two-level online optimization to generate customized accelerators for ASIC and FPAG platforms. In contrast, PRIME , does not only learn a conservative model of the objective function from offline data but can also leverage information from infeasible points and can work with just a few thousand feasible points. In addition, we devise a contextual version of PRIME that is effective in designing accelerators that are jointly optimized for multiple applications, different from prior work. Finally, to our

knowledge, our work, is the first to demonstrate generalization to unseen applications for accelerator design, outperforming state-of-the-art online methods.

A popular approach for solving black-box optimization problems is model-based optimization (MBO) [305, 292, 304]. Most of the classical MBO methods fail to scale to high-dimensions, and have been extended with neural networks [305, 304, 168, 101, 100, 12, 11, 229]. While these methods work well in the active setting, they are susceptible to out-of-distribution inputs [327] in the offline, data-driven setting. To prevent this, many methods for offline model-based optimization constrain the optimizer to the manifold of valid, in-distribution inputs [33, 78, 178]. However, modeling the manifold of valid inputs can be challenging for accelerators. PRIME dispenses with the need for generative modeling, while still avoiding out-of-distribution inputs. PRIME takes a conservative value estimation approach. However, unlike these approaches, PRIME can handle constraints by learning from infeasible data. In addition, while prior works in this area mostly restricted their design problem to a single application, we show that PRIME is effective for multi-application optimization and zero-shot generalization.

10.6 Experimental Evaluation

Our evaluations aim to answer the following questions: **Q(1)** Can PRIME design accelerators tailored for a given application that are better than the best observed configuration in the training dataset, and comparable to or better than state-of-the-art simulation-driven methods under a given simulator-query budget? **Q(2)** Does PRIME reduce the total simulation time compared to other methods? **Q(3)** Can PRIME produce hardware accelerators for a family of different applications? **Q(4)** Can PRIME trained for a family of applications extrapolate to designing a high-performing accelerator for a new, unseen application, thereby enabling data reuse? Additionally, we ablate various properties of PRIME (Appendix H.1.6) and evaluate its efficacy in designing accelerators with distinct dataflow architectures and a larger search space (up to 2.5×10^{114} candidates).

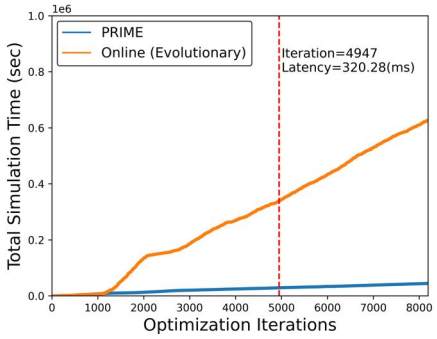


Figure 14: Comparing the total simulation time of PRIME (for PRIME this is the total time for a forward-pass through the trained model on a CPU) and evolutionary method on MobileNetEdgeTPU. PRIME only requires about **7%** of the total simulation time of the online method.

Baselines and comparisons. We compare PRIME against three online optimization methods that actively query the simulator: **(1)** evolutionary search with the firefly optimizer [360] ("Evolutionary"), which is the shown to outperform other online methods for accelerator design; **(2)** Bayesian Optimization ("Bayes Opt") [108], **(3)** MBO [11]. In all the experiments, we grant all the methods the same number of feasible points. Note

Table 30: Optimized objective values (i.e., latency in milliseconds) obtained by various methods for the task of learning accelerators specialized to a given application. Lower latency is better. **From left to right**: our method, online Bayesian optimization ("Bayes Opt"), online evolutionary algorithm ("Evolutionary"), and the best design in the training dataset. On average (last row), Cal-QL improves over the best in the dataset by 2.46× (up to 6.69× in t-RNN Dec) and outperforms best online optimization methods by 1.54× (up to 6.62× in t-RNN Enc). The best accelerator configurations identified is highlighted in bold.

Application	PRIME	Online Optimization			$\mathcal{D}$ (Best in Training)
		Bayes Opt	Evolutionary	MBO	
MobileNetEdgeTPU	**298.50**	319.00	320.28	332.97	354.13
MobileNetV2	**207.43**	240.56	238.58	244.98	410.83
MobileNetV3	**454.30**	534.15	501.27	535.34	938.41
M4	**370.45**	396.36	383.58	405.60	779.98
M5	208.21	201.59	**198.86**	219.53	449.38
M6	131.46	121.83	120.49	**119.56**	369.85
U-Net	**740.27**	872.23	791.64	888.16	1333.18
t-RNN Dec	**132.88**	771.11	770.93	771.70	890.22
t-RNN Enc	**130.67**	865.07	865.07	866.28	584.70
Geomean of PRIME 's Improvement	1.0×	1.58×	1.54×	1.61×	2.46×

that our method do not get to select these points, and use the same exact offline points across all the runs, while the online methods can actively select which points to query, and therefore require new queries for every run. "$\mathcal{D}$(Best in Training)" denotes the best latency value in the training dataset used in PRIME . We also present ablation results with different components of our method removed in Appendix H.1.6, where we observe that utilizing both infeasible points and negative sampling are generally important for attaining good results. Appendix H.1.1 presents additional comparisons P3BO [12], a state-of-the-art method studied in the context of biological sequence design.

Architecting application-specific accelerators. We first evaluate PRIME in designing specialized accelerators for each of the applications in Table 29. We train a conservative model using the method in Section 10.4 on the logged dataset for each application separately. The area constraint α (Equation 10.3.1) is set to $\alpha = 29$ mm^2, a realistic budget for accelerators [360]. Table 30 summarizes the results. On average, the best accelerators designed by PRIME outperforms the best accelerator configuration in the training dataset (last row Table 30), by 2.46×. PRIME also outperforms the accelerators in the best online method by 1.54× (up to 5.80× and 6.62× in t-RNN Dec and t-RNN Enc, respectively). Moreover, perhaps surprisingly, PRIME generates accelerators that are better than all the online optimization methods in 7/9 domains, and performs on par in several other scenarios (on average only 6.8% slowdown compared to the best accelerator with online methods in M5 and M6). These results indicates that offline optimization of accelerators using PRIME can be more data-efficient compared to online methods with active simulation queries. To answer **Q(2)**, we compare the total simulation time of

Table 31: Optimized average latency (the lower, the better) across multiple applications (up to ten applications) for PRIME and best online algorithms (Evolutionary, MBO). Each row show the (Best, Median) of average latency across five runs. The geometric mean of PRIME 's improvement over other methods (last row) indicates that PRIME is at least 21% better.

Applications	Area	PRIME (Ours)	Evolutionary (Online)	MBO (Online)
MobileNet (EdgeTPU, V2, V3)	29 mm^2	(310.21, 334.70)	**(315.72, 325.69)**	(342.02, 351.92)
MobileNet (V2, V3), M5, M6	29 mm^2	**(268.47, 271.25)**	(288.67, 288.68)	(295.21, 307.09)
MobileNet (EdgeTPU, V2, V3), M4, M5, M6	29 mm^2	**(311.39, 313.76)**	(314.31, 316.65)	(321.48, 339.27)
MobileNet (EdgeTPU, V2, V3), M4, M5, M6, U-Net, t-RNN-Enc	29 mm^2	**(305.47, 310.09)**	(404.06, 404.59)	(404.06, 412.90)
MobileNet (EdgeTPU, V2, V3), M4, M5, M6, t-RNN-Enc	100 mm^2	**(286.45, 287.98)**	(404.25, 404.59)	(404.06, 404.94)
MobileNet (EdgeTPU, V2, V3), M4, M5, M6, t-RNN (Dec, Enc)	29 mm^2	**(426.65, 426.65)**	(586.55, 586.55)	(626.62, 692.61)
MobileNet (EdgeTPU, V2, V3), M4, M5, M6, U-Net, t-RNN (Dec, Enc)	100 mm^2	**(383.57, 385.56)**	(518.58, 519.37)	(526.37, 530.99)
Geomean of PRIME 's Improvement	—	(1.0×, 1.0×)	(1.21×, 1.20×)	(1.24×, 1.27×)

PRIME and the best evolutionary approach from Table 30 on the MobileNetEdgeTPU domain. On average, not only that PRIME outperforms the best online method that we evaluate, but also considerably reduces the total simulation time by 93%, as shown in Figure 14. Even the total simulation time to the first occurrence of the final design that is eventually returned by the online methods is about 11× what PRIME requires to fine a better design. This indicates that data-driven PRIME is much more preferred in terms of the performance-time trade-off. The fact that our offline approach PRIME outperforms the online evolutionary method (and also other state-of-the-art online MBO methods; see Table 70) is surprising, and we suspect this is because online methods get stuck early-on during optimization, while utilizing offline data allows us PRIME to find better solutions via generalization (see Appendix H.2.1.1).

Architecting accelerators for multiple applications. To answer **Q(3)**, we evaluate the efficacy of the contextual version of PRIME in designing an accelerator that attains the lowest latency averaged over a set of application domains. As discussed previously, the training data used does not label a given accelerator with latency values corresponding to each application, and thus, PRIME must extrapolate accurately to estimate the latency of an accelerator for a context it is not paired with in the training dataset. This also means that PRIME cannot simply return the accelerator with the best average latency and must run non-trivial optimization. We evaluate our method in seven different scenarios (Table 31), comprising various combinations of models from Table 29 and under different area constraints, where the smallest set consists of the three MobileNet variants and the largest set consists of nine models from image classification, object detection, image segmentation,

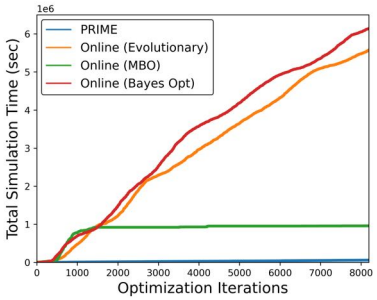


Figure 15: Comparing the total simulation time needed by PRIME and online methods on seven models (Area $\leq$ 100mm^2) . PRIME only requires about 1%, 6%, and 0.9% of the total simulation time of Evolutionary, MBO, and Bayes Opt, respectively, although PRIME outperforms the best online method by 41%.

and speech recognition. This scenario is also especially challenging for online methods since the number of jointly feasible designs is expected to drop significantly as more applications are added. For instance, for the case of the MobileNet variants, the training dataset only consists of a few (20-30) accelerator configurations that are jointly feasible and high-performing (Appendix H.2.2—Figure 54).

Table 31 shows that, on average, PRIME finds accelerators that outperform the best online method by 1.2× (up to 41%). While PRIME performs similar to online methods in the smallest three-model scenario (first row), it outperforms online methods as the number of applications increases and the set of applications become more diverse. In addition, comparing with the best jointly feasible design point across the target applications, PRIME finds significantly better accelerators (3.95×). Finally, as the number of model increases the total simulation time difference between online methods and PRIME further widens (Figure 15). These results indicate that PRIME is effective in designing accelerators jointly optimized across multiple applications while reusing the same dataset as for the single-task, and scales more favorably than its simulation-driven counterparts. Appendix H.1.4 expounds the details of the designed accelerators for nine applications, comparing our method and the best online method.

Accelerating previously unseen applications ("zero-shot" optimization). Finally, we answer **Q(4)** by demonstrating that our data-driven offline method, PRIME enables effective data reuse by using logged accelerator data from a set of applications to design an accelerator for an unseen new application, without requiring any training on data from the new unseen application(s). We train a contextual version of PRIME using a set of "training applications" and then optimize an accelerator using the learned model with different contexts corresponding to "test applications," without any additional query to the test application dataset. Table 32 shows, on average, PRIME outperforms the best online method by 1.26× (up to 66%) and only 2% slowdown in 1/4 cases. Note that the difference in performance increases as the number of training applications increases. These results show the effectiveness of PRIME in the zero-shot setting (more results in Appendix H.1.5).

Applying PRIME on other accelerator architectures and dataflows. Finally, to assess the the generalizability of PRIME to other accelerator architectures [164], we evaluate PRIME to optimize latency of two style of dataflow accelerators—NVDLA-style and ShiDianNao-style—across three applications (Appendix H.3 details the methodology). As shown in Table 33, PRIME outperforms the online evolutionary method by 6% and improves over the best point in the training dataset by 3.75×. This demonstrates the efficacy of PRIME with different dataflows and large design spaces.

Table 32: Optimized objective values (i.e., latency in milliseconds) under zero-shot setting. Lower latency is better. From left to right: the applications used to train the surrogate model in Cal-QL the target applications for which the accelerator is optimized for, the area constraint of the accelerator, Cal-QL's (best, median) latency, and best online method's (best, median) latency. Cal-QL does not use any additional data from the target applications. On average (last row), Cal-QL yields optimized accelerator for target applications (with zero query to the target applications' dataset) with 1.26× (up to 1.66×) lower latency over the best online method. The best accelerator configurations identified is highlighted in bold.

Train Applications	Test Applications	Area	PRIME (Ours)	Evolutionary (Online)
MobileNet (EdgeTPU, V3)	MobileNetV2	29 mm^2	(**311.39, 313.76**)	(314.31, 316.65)
MobileNet (V2, V3), M5, M6	MobileNetEdge, M4	29 mm^2	(357.05, 364.92)	(**354.59, 357.29**)
MobileNet (EdgeTPU, V2, V3), M4, M5, M6, t-RNN Enc	U-Net, t-RNN Dec	29 mm^2	(**745.87, 745.91**)	(1075.91, 1127.64)
MobileNet (EdgeTPU, V2, V3),M4, M5, M6, t-RNN Enc	U-Net, t-RNN Dec	100 mm^2	(**517.76, 517.89**)	(859.76, 861.69)
Geomean of PRIME 's Improvement	—	—	(1.0×, 1.0×)	(1.24×, 1.26×)

Table 33: Optimized objective values (i.e. total number of cycles) for two different dataflow architectures, NVDLA-style [249] and ShiDianNao-style [67], across three classes of applications. The maximum search space for the studied accelerators are $\approx 2.5\times10^{114}$. PRIME generalizes to other classes of accelerators with larger search space and outperforms the best online method by 1.06× and the best data seen in training by 3.75× (last column). The best accelerator configurations is highlighted in bold.

Applications	Dataflow	PRIME	Evolutionary (Online)	$\mathcal{D}$ (Best in Training)
MobileNetV2	NVDLA	$\mathbf{2.51\times10^7}$	2.70×10^7	1.32×10^8
MobileNetV2	ShiDianNao	$\mathbf{2.65\times10^7}$	2.84×10^7	1.27×10^8
ResNet50	NVDLA	$\mathbf{2.83\times10^8}$	3.13×10^8	1.63×10^9
ResNet50	ShiDianNao	$\mathbf{3.44\times10^8}$	3.74×10^8	2.05×10^9
Transformer	NVDLA	$\mathbf{7.8\times10^8}$	$\mathbf{7.8\times10^8}$	1.3×10^9
Transformer	ShiDianNao	$\mathbf{7.8\times10^8}$	$\mathbf{7.8\times10^8}$	1.5×10^9
Geomean of PRIME 's Improvement	—	1.0×	1.06×	3.75×

10.7 DISCUSSION

In this work, we present a data-driven offline optimization method, PRIME to automatically architect hardware accelerators. Our method learns a conservative model of the objective function by leveraging infeasible data points to better model the desired objective function of the accelerator using a one-time collected dataset of accelerators, thereby alleviating the need for time-consuming simulation. Our results show that, on average, our method outperforms the best designs observed in the logged data by 2.46× and improves over the best simulator-driven approach by about 1.54×. In the more challenging setting of designing accelerators jointly optimal for multiple applications or for new, unseen applications, zero-shot, PRIME outperforms simulator-driven methods by 1.2×, while reducing the total simulation time by 99%. The efficacy of PRIME highlights the potential for utilizing the logged offline data in an accelerator design pipeline. While PRIME outperforms the online methods we utilize, in principle, a strong online method can be devised by running PRIME in the inner loop. Our goal is to not advocate

that offline methods must replace online methods, but that training a strong offline optimization algorithm on offline datasets of low-performing designs can be a highly effective ingredient in hardware accelerator design.

ACKNOWLEDGEMENTS AND FUNDING

We thank the "Learn to Design Accelerators" and EdgeTPU teams at Google for their invaluable feedback. In addition, we extend our gratitude to the Vizier team, Christof Angermueller, Sheng-Chun Kao, Samira Khan, Stella Aslibekyan, and Xinyang Geng for their help with experiment setups and insightful comments.

11

CONCLUSION

Over the course of various chapters in this dissertation, we have attempted to develop a "tool-box" for utilizing static datasets in reinforcement learning. In Part I of this thesis, we began by understanding the challenges in the offline RL setting by running diagnostic experiments in a "unit-testing" framework, followed by developing an initial offline RL approach, BEAR, which restricted policy learning to the support of the offline data in Part II. Refining this approach, we then developed the paradigm of conservative value estimation and instantiated it into two algorithms: model-free CQL and model-based COMBO. We found that CQL and COMBO attain state-of-the-art empirical results. The efficacy of this paradigm is reflected in the use of CQL by external practitioners to tackle a variety of real-world decision-making problems.

Building on the foundations provided by the principle of conservative value estimation, in Part III of this dissertation we developed approaches that make it possible to utilize offline RL effectively with high-capacity models and diverse datasets. We saw how the implicit regularization of stochastic gradient descent, which is widely believed to aid generalization of over-parameterized deep neural networks in supervised learning, can hurt when learning value functions in the offline setting. Building on this insight, we proposed the explicit DR3 regularizer that alleviates pathologies due to implicit regularization in offline RL and showed that in conjunction with the ResNet architectures, DR3 enables offline Q-learning to scale well to high-capacity models. Then, we turned our attention to the problem of maximally utilizing diverse data and proposed the CDS approach for intelligently relabeling data in multi-task offline RL problems and extended it via the UDS approach to deal with scenarios when reward annotations are hard to obtain during data sharing. We also developed Cal-QL, a method that extends CQL to make it effective at fast online fine-tuning, after offline training. Finally, in Part IV of this dissertation, we utilized the aforementioned algorithmic advancements to tackle decision-making problems in two application domains.

OPEN PROBLEMS. Zooming out, the tools discussed in this dissertation and several other works in the field of RL with static datasets provide an effective starting point

for **building broadly deployable decision-making systems**, but there are still several questions that I believe are important to address towards attaining this goal. Here, I discuss some questions that I especially find the most compelling to address.

Foundation models for decision-making that use multi-modal data. The mantra in modern ML is that larger and more diverse the dataset, the better the performance. However, problem-specific datasets for many sequential decision-making problems (e.g., protein design, human-AI interaction) are much smaller and narrow as well. We can attempt to tackle this issue by incorporating multi-modal datasets from other sources (e.g., language annotations of proteins for protein design). But this is tricky as it requires a departure from the standard RL paradigm since multi-modal datasets often do not satisfy several pre-conditions needed for RL, e.g., no transitions, no rewards, no sequential structure. It is also unclear if ways to incorporate multi-modal data from supervised learning alone are sufficient for decision-making. Some of our initial works [366, 368, 9], including UDS from Chapter 7 take initial steps towards this goal, but nonetheless, it is important for future work to study questions pertaining to extracting rewards, representations and structures from multi-modal data using insights from self-supervised learning and pre-training. Perhaps a particularly interesting initial step in the current research atmosphere is developing reinforcement learning methods that can utilize initializations provided by multi-modal pre-trained foundation models.

Understanding and building reliable and easy-to-tune RL methods. One of the major reasons behind the widespread adoption of supervised and unsupervised ML in real-world systems is that these methods behave consistently and are easy to tune, greatly reducing the need to deploy imperfect models. To enable a broad real-world deployment of RL, we need to ensure RL enjoys similar benefits. This entails a continued effort on understanding and analyzing the failures and learning dynamics of offline RL building on our insights in Chapter 6; designing theoretically sound recipes for tuning design decisions in offline RL, equivalently to cross-validation in standard machine learning; understanding which RL algorithms must be used when the underlying problem satisfies specific certain structural conditions; to list a few. Some of my work [184, 186] tackles questions of this sort using tools from optimization and statistical machine learning, but a complete solution is still missing and a promising avenuew for future work. I believe that the next generation of reinforcement learning algorithms will be methods that come equipped with protocols or "workflows" for practitioners to use them effectively. Ultimately, success towards this goal comes down to the usability of techniques in the real world, and hence we believe that it would be especially important to study these questions while being grounded in real-world applications.

Robustness and adaptability in the rapidly-changing real world. Upon deployment, RL algorithms that train on static datasets must not just run the learned policy, but they must

also perceive and adapt to the current state of the real world (for e.g., understand the intent of users around them) while exhibiting safe and robust behavior. They also need to tackle implicit biases and heterogeneity in experience they will gather. For example, a drug discovery method must learn to neutralize a new pathogen in the least amount of trials while being safe for consumption at every step in the process; a recommender system should not promote unethical content and amplify biases in society just to attain a higher click-through-rate. This presents a challenge as we need to develop offline RL algorithms that not only leverage static datasets to learn policies, but also to learn strategies that are adaptive and exhibit safe behavior. One potential avenue to study these questions could be to build on my initial work [28, 9] and utilize insights from areas such as robustness, adaptation, and meta learning.

Advancing real-world applications. An important, but overlooked factor in RL algorithms research is iterating on real-world problems alongside algorithmic research. In general, the RL problem is often intractable, and indeed, this is evident from a number of theoretical results showing hardness of RL. This also means that making long-term progress in RL needs to be grounded on a set of real-world problems, with representative properties. Some promising avenues for future work includes problems in **computer architecture** [81], **robotic manipulation**, and also problems understudied in the context of RL such as **optimizing drug designs**, which requires building easy-to-tune algorithms capable of learning from multi-modal datasets. Another interesting problem in modern-day context is that of fine-tuning deep learning models to make them amenable for safe interaction with humans (**human-AI interaction**), which requires easy-to-use RL algorithms that can leverage static data while also exhibiting rapid adaptation. Another example is the problem of making decisions in **electrical systems** (e.g., power grids, HVAC systems [340]) to control climate change, using historical data, safe and robustly deployable methods. It would also be very valuable to bring the insights from these domains back to the algorithms community by building benchmarks that serve as milestones to guide research in the years to come.

BIBLIOGRAPHY

[1] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *ICML*, 2004.

[2] Abbas Abdolmaleki, Jost Tobias Springenberg, Y. Tassa, R. Munos, N. Heess, and Martin A. Riedmiller. Maximum a posteriori policy optimisation. *ArXiv*, abs/1806.06920, 2018.

[3] Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 22–31. JMLR. org, 2017.

[4] Joshua Achiam, Ethan Knight, and Pieter Abbeel. Towards characterizing divergence in deep q-learning. *arXiv preprint arXiv:1903.08894*, 2019.

[5] Alekh Agarwal, Nan Jiang, and Sham M Kakade. Reinforcement learning: Theory and algorithms. *CS Dept., UW Seattle, Seattle, WA, USA, Tech. Rep*, 2019.

[6] Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. An optimistic perspective on offline reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2020.

[7] Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron Courville, and Marc G Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *Advances in Neural Information Processing Systems*, 2021.

[8] Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C Courville, and Marc Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *Advances in Neural Information Processing Systems*, 34, 2021.

[9] Anurag Ajay, Aviral Kumar, Pulkit Agrawal, Sergey Levine, and Ofir Nachum. Opal: Offline primitive discovery for accelerating offline reinforcement learning. *arXiv preprint arXiv:2010.13611*, 2020.

[10] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. *arXiv preprint arXiv:1707.01495*, 2017.

[11] Christof Angermueller, David Dohan, David Belanger, Ramya Deshpande, Kevin Murphy, and Lucy Colwell. Model-based Reinforcement Learning for Biological Sequence Design. In *ICLR*, 2019.

[12] Christof Angermueller, David Belanger, Andreea Gane, Zelda Mariet, David Dohan, Kevin Murphy, Lucy Colwell, and D Sculley. Population-Based Black-Box Optimization for Biological Sequence Design. In *ICML*, 2020.

[13] Jason Ansel, Shoaib Kamil, Kalyan Veeramachaneni, Jonathan Ragan-Kelley, Jeffrey Bosboom, Una-May O'Reilly, and Saman Amarasinghe. OpenTuner: An Extensible Framework for Program Autotuning. In *PACT*, 2014.

[14] Andräs Antos, Csaba Szepesvari, and Remi Munos. Value-iteration based fitted policy iteration: Learning with a single trajectory. In *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pages 330–337, April 2007. doi: 10.1109/ADPRL.2007.368207.

[15] András Antos, Csaba Szepesvári, and Rémi Munos. Fitted q-iteration in continuous action-space mdps. In *Advances in Neural Information Processing Systems 20*, pages 9–16. Curran Associates, Inc., 2008.

[16] Arthur Argenson and Gabriel Dulac-Arnold. Model-based offline planning. *arXiv preprint arXiv:2008.05556*, 2020.

[17] Sanjeev Arora, Nadav Cohen, and Elad Hazan. On the optimization of deep networks: Implicit acceleration by overparameterization. *arXiv preprint arXiv:1802.06509*, 2018.

[18] A Badia, B Piot, S Kapturowski, P Sprechmann, A Vitvitskyi, D Guo, and C Blundell. Agent57: Outperforming the human atari benchmark. In *ICML*, 2020.

[19] Prasanna Balaprakash, Ananta Tiwari, Stefan M Wild, Laura Carrington, and Paul D Hovland. AutoMOMML: Automatic Multi-Objective Modeling with Machine Learning. In *HiPC*, 2016.

[20] Philip J Ball, Laura Smith, Ilya Kostrikov, and Sergey Levine. Efficient online reinforcement learning with offline data. *arXiv preprint arXiv:2302.02948*, 2023.

[21] Alex Beeson and Giovanni Montana. Improving TD3-BC: Relaxed Policy Constraint for Offline Learning and Stable Online Fine-Tuning. *arXiv arXiv:2211.11802*, 2022.

[22] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[23] Marc G Bellemare, Georg Ostrovski, Arthur Guez, Philip S Thomas, and Rémi Munos. Increasing the action gap: New operators for reinforcement learning. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[24] Marc G Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 449–458. JMLR. org, 2017.

[25] Richard Bellman. Dynamic programming. *Science*, 153(3731):34–37, 1966.

[26] Emmanuel Bengio, Joelle Pineau, and Doina Precup. Interference and generalization in temporal difference learning. *arXiv preprint arXiv:2003.06350*, 2020.

[27] Dimitri P Bertsekas and John N Tsitsiklis. *Neuro-dynamic programming*. Athena Scientific, 1996.

[28] Homanga Bharadhwaj, Aviral Kumar, Nicholas Rhinehart, Sergey Levine, Florian Shkurti, and Animesh Garg. Conservative safety critics for exploration. *arXiv preprint arXiv:2010.14497*, 2020.

[29] Aditya Bhatt, Max Argus, Artemij Amiranashvili, and Thomas Brox. CrossNorm: Normalization for Off-Policy TD Reinforcement Learning. *arXiv e-prints*, art. arXiv:1902.05605, February 2019.

[30] Johan Bjorck, Carla P Gomes, and Kilian Q Weinberger. Towards deeper deep reinforcement learning. *arXiv preprint arXiv:2106.01151*, 2021.

[31] Guy Blanc, Neha Gupta, Gregory Valiant, and Paul Valiant. Implicit regularization for deep neural networks driven by an ornstein-uhlenbeck like process. In *Conference on learning theory*, pages 483–513. PMLR, 2020.

[32] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[33] David Brookes, Hahnbeom Park, and Jennifer Listgarten. Conditioning by Adaptive Sampling for Robust Design. In *ICML*, 2019.

[34] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[35] Jacob Buckman, Carles Gelada, and Marc G Bellemare. The importance of pessimism in fixed-dataset policy optimization. *arXiv preprint arXiv:2009.06799*, 2020.

[36] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*, 2018.

[37] Rich Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997.

[38] Pablo Samuel Castro, Subhodeep Moitra, Carles Gelada, Saurabh Kumar, and Marc G Bellemare. Dopamine: A research framework for deep reinforcement learning. *arXiv preprint arXiv:1812.06110*, 2018.

[39] Prasanth Chatarasi, Hyoukjun Kwon, Natesh Raina, Saurabh Malik, Vaisakh Haridas, Tushar Krishna, and Vivek Sarkar. MARVEL: A Decoupled Model-driven Approach for Efficiently Mapping Convolutions on Spatial DNN Accelerators. *arXiv preprint arXiv:2002.07752*, 2020.

[40] Niladri S Chatterji, Behnam Neyshabur, and Hanie Sedghi. The intriguing role of module criticality in the generalization of deep networks. *arXiv preprint arXiv:1912.00528*, 2019.

[41] Yevgen Chebotar, Karol Hausman, Yao Lu, Ted Xiao, Dmitry Kalashnikov, Jake Varley, Alex Irpan, Benjamin Eysenbach, Ryan Julian, Chelsea Finn, et al. Actionable models: Unsupervised offline reinforcement learning of robotic skills. *arXiv preprint arXiv:2104.07749*, 2021.

[42] Annie S Chen, Suraj Nair, and Chelsea Finn. Learning generalizable robotic reward functions from" in-the-wild" human videos. *arXiv preprint arXiv:2103.16817*, 2021.

[43] Jinglin Chen and Nan Jiang. Information-theoretic considerations in batch reinforcement learning. *arXiv preprint arXiv:1905.00360*, 2019.

[44] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *NeurIPS*, 2021.

[45] Xinlei Chen and Kaiming He. Exploring simple siamese representation learning. *arXiv preprint arXiv:2011.10566*, 2020.

[46] Xinyue Chen, Che Wang, Zijian Zhou, and Keith W. Ross. Randomized ensembled double q-learning: Learning fast without a model. In *International Conference on Learning Representations*, 2021.

[47] Ching-An Cheng, Tengyang Xie, Nan Jiang, and Alekh Agarwal. Adversarially trained actor critic for offline reinforcement learning. *ICML*, 2022.

[48] Paul Christiano, Jan Leike, Tom B Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *arXiv preprint arXiv:1706.03741*, 2017.

[49] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Advances in Neural Information Processing Systems*, pages 4754–4765, 2018.

[50] Jason Cong, Peng Wei, Cody Hao Yu, and Peng Zhang. Automated Accelerator Generation and Optimization with Composable, Parallel and Pipeline Architecture. In *DAC*, 2018.

[51] Will Dabney, Mark Rowland, Marc G Bellemare, and Rémi Munos. Distributional reinforcement learning with quantile regression. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[52] Alex Damian, Tengyu Ma, and Jason Lee. Label noise sgd provably prefers flat global minimizers. *arXiv preprint arXiv:2106.06530*, 2021.

[53] Sudeep Dasari, Frederik Ebert, Stephen Tian, Suraj Nair, Bernadette Bucher, Karl Schmeckpeper, Siddharth Singh, Sergey Levine, and Chelsea Finn. Robonet: Large-scale multi-robot learning. In *Conference on Robot Learning*, 2020.

[54] Shail Dave, Youngbin Kim, Sasikanth Avancha, Kyoungwoo Lee, and Aviral Shrivastava. DMazeRunner: Executing Perfectly Nested Loops on Dataflow Accelerators. *TECS*, 2019.

[55] Tim de Bruin, Jens Kober, Karl Tuyls, and Robert Babuska. The importance of experience replay database composition in deep reinforcement learning. 01 2015.

[56] Daniela Pucci De Farias. *The linear programming approach to approximate dynamic programming: Theory and application*. PhD thesis, 2002.

[57] Victoria Dean, Daniel Kenji Toyama, and Doina Precup. Don't freeze your embedding: Lessons from policy finetuning in environment transfer. In *ICLR Workshop on Agent Learning in Open-Endedness*, 2022.

[58] Thomas Degris, Martha White, and Richard S Sutton. Off-policy actor-critic. *arXiv preprint arXiv:1205.4839*, 2012.

[59] Mostafa Dehghani, Alexey Gritsenko, Anurag Arnab, Matthias Minderer, and Yi Tay. Scenic: A jax library for computer vision research and beyond. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 21393–21398, 2022.

[60] Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *International Conference on machine learning (ICML)*, pages 465–472, 2011.

[61] Carlo D'Eramo, Davide Tateo, Andrea Bonarini, Marcello Restelli, and Jan Peters. Sharing knowledge in multi-task deep reinforcement learning. In *International Conference on Learning Representations*, 2019.

[62] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[63] Ron Dorfman and Aviv Tamar. Offline meta reinforcement learning. *arXiv e-prints*, pages arXiv–2008, 2020.

[64] Pierluca D'Oro, Max Schwarzer, Evgenii Nikishin, Pierre-Luc Bacon, Marc G Bellemare, and Aaron Courville. Sample-efficient reinforcement learning by breaking the replay ratio barrier. In *The Eleventh International Conference on Learning Representations*, 2023.

[65] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[66] Simon Du, Sham Kakade, Jason Lee, Shachar Lovett, Gaurav Mahajan, Wen Sun, and Ruosong Wang. Bilinear classes: A structural framework for provable generalization in rl. In *International Conference on Machine Learning*, pages 2826–2836. PMLR, 2021.

[67] Zidong Du, Robert Fasthuber, Tianshi Chen, Paolo Ienne, Ling Li, Tao Luo, Xiaobing Feng, Yunji Chen, and Olivier Temam. ShiDianNao: Shifting Vision Processing Closer to the Sensor. In *ISCA*, 2015.

[68] Yaqi Duan, Zeyu Jia, and Mengdi Wang. Minimax-optimal off-policy evaluation with linear function approximation. In *International Conference on Machine Learning*, pages 2701–2709. PMLR, 2020.

[69] Ishan Durugkar and Peter Stone. Td learning with constrained gradients. 2018.

[70] Frederik Ebert, Yanlai Yang, Karl Schmeckpeper, Bernadette Bucher, Georgios Georgakis, Kostas Daniilidis, Chelsea Finn, and Sergey Levine. Bridge data: Boosting generalization of robotic skills with cross-domain datasets. *arXiv preprint arXiv:2109.13396*, 2021.

[71] Scott Emmons, Benjamin Eysenbach, Ilya Kostrikov, and Sergey Levine. Rvs: What is essential for offline rl via supervised learning? *arXiv preprint arXiv:2112.10751*, 2021.

[72] Damien Ernst, Pierre Geurts, and Louis Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6(Apr):503–556, 2005.

[73] Hadi Esmaeilzadeh, Emily Blem, Renee St Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark Silicon and the End of Multicore Scaling. In *ISCA*, 2011.

[74] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. *arXiv preprint arXiv:1802.01561*, 2018.

[75] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2018.

[76] Ben Eysenbach, Xinyang Geng, Sergey Levine, and Russ R Salakhutdinov. Rewriting history with inverse rl: Hindsight inference for policy improvement. *Advances in neural information processing systems*, 33:14783–14795, 2020.

[77] Benjamin Eysenbach, Sergey Levine, and Ruslan Salakhutdinov. Replacing rewards with examples: Example-based policy search via recursive classification. *arXiv preprint arXiv:2103.12656*, 2021.

[78] Clara Fannjiang and Jennifer Listgarten. Autofocused Oracles for Model-Based Design. *arXiv preprint arXiv:2006.08052*, 2020.

[79] Amir-massoud Farahmand, Csaba Szepesvári, and Rémi Munos. Error propagation for approximate policy and value iteration. In *Advances in Neural Information Processing Systems*, pages 568–576, 2010.

[80] Jesse Farebrother, Marlos C Machado, and Michael Bowling. Generalization and regularization in dqn. *arXiv preprint arXiv:1810.00123*, 2018.

[81] Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Francisco J R Ruiz, Julian Schrittwieser, Grzegorz Swirszcz, et al. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610(7930):47–53, 2022.

[82] William Fedus, Prajit Ramachandran, Rishabh Agarwal, Yoshua Bengio, Hugo Larochelle, Mark Rowland, and Will Dabney. Revisiting fundamentals of experience replay. *arXiv preprint arXiv:2007.06700*, 2020.

[83] Chelsea Finn and Sergey Levine. Deep visual foresight for planning robot motion. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2786–2793. IEEE, 2017.

[84] Chelsea Finn, Sergey Levine, and Pieter Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *International conference on machine learning*, pages 49–58. PMLR, 2016.

[85] Michael J Flynn and Wayne Luk. *Computer System Design: System-on-Chip*. John Wiley & Sons, 2011.

[86] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine. D4rl: Datasets for deep data-driven reinforcement learning. In *arXiv*, 2020. URL https://arxiv.org/pdf/2004.07219.

[87] Justin Fu and Sergey Levine. Offline Model-Based Optimization via Normalized Maximum Likelihood Estimation. In *ICLR*, 2021.

[88] Justin Fu, Katie Luo, and Sergey Levine. Learning robust rewards with adversarial inverse reinforcement learning. *International Conference on Learning Representations*, 2018.

[89] Justin Fu, Avi Singh, Dibya Ghosh, Larry Yang, and Sergey Levine. Variational inverse control with events: A general framework for data-driven reward definition. *NeurIPS*, 2018.

[90] Justin Fu, Aviral Kumar, Matthew Soh, and Sergey Levine. Diagnosing bottlenecks in deep q-learning algorithms. *arXiv preprint arXiv:1902.10250*, 2019.

[91] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for data-driven deep reinforcement learning. https://github.com/rail-berkeley/d4rl/wiki/New-Franka-Kitchen-Tasks, 2020. Github repository.

[92] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

[93] Justin Fu, Mohammad Norouzi, Ofir Nachum, George Tucker, ziyu wang, Alexander Novikov, Mengjiao Yang, Michael R Zhang, Yutian Chen, Aviral Kumar, Cosmin Paduraru, Sergey Levine, and Thomas Paine. Benchmarks for deep off-policy evaluation. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=kWSeGEeHvF8.

[94] Scott Fujimoto and Shixiang Shane Gu. A minimalist approach to offline reinforcement learning. *arXiv preprint arXiv:2106.06860*, 2021.

[95] Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. *arXiv preprint arXiv:1812.02900*, 2018.

[96] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1587–1596. PMLR, 2018.

[97] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning (ICML)*, pages 1587–1596, 2018.

[98] Scott Fujimoto, Edoardo Conti, Mohammad Ghavamzadeh, and Joelle Pineau. Benchmarking batch deep reinforcement learning algorithms. *arXiv preprint arXiv:1910.01708*, 2019.

[99] Yang Gao, Huazhe Xu, Ji Lin, Fisher Yu, Sergey Levine, and Trevor Darrell. Reinforcement learning from imperfect demonstrations. In *ICLR (Workshop)*. OpenReview.net, 2018.

[100] Marta Garnelo, Dan Rosenbaum, Christopher Maddison, Tiago Ramalho, David Saxton, Murray Shanahan, Yee Whye Teh, Danilo Rezende, and S. M. Ali Eslami. Conditional Neural Processes. In *ICML*, 2018.

[101] Marta Garnelo, Jonathan Schwarz, Dan Rosenbaum, Fabio Viola, Danilo J. Rezende, S. M. Ali Eslami, and Yee Whye Teh. Neural Processes. *arXiv preprint arXiv:1807.01622*, 2018.

[102] Carles Gelada and Marc G. Bellemare. Off-policy deep reinforcement learning by bootstrapping the covariate shift. *CoRR*, abs/1901.09455, 2019.

[103] Michael A Gelbart, Jasper Snoek, and Ryan P Adams. Bayesian Optimization with Unknown Constraints. *arXiv preprint arXiv:1403.5607*, 2014.

[104] Xinyang Geng. Jaxcql: a simple implementation of sac and cql in jax. 2022. URL https://github.com/young-geng/JaxCQL.

[105] Seyed Kamyar Seyed Ghasemipour, Dale Schuurmans, and Shixiang Shane Gu. Emaq: Expected-max q-learning operator for simple yet effective offline and online rl. In *International Conference on Machine Learning*, pages 3682–3691. PMLR, 2021.

[106] Dibya Ghosh and Marc G Bellemare. Representations for stable off-policy reinforcement learning. *arXiv preprint arXiv:2007.05520*, 2020.

[107] Dibya Ghosh, Avi Singh, Aravind Rajeswaran, Vikash Kumar, and Sergey Levine. Divide-and-conquer reinforcement learning. *arXiv preprint arXiv:1711.09874*, 2017.

[108] Daniel Golovin, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, and D Sculley. Google Vizier: A Service for Black-box Optimization. In *SIGKDD*, 2017.

[109] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and Harnessing Adversarial Examples. *ICLR*, 2015.

[110] Robert J Gove, Keith Balmer, Nicholas K Ing-Simmons, and Karl M Guttag. Multi-Processor Reconfigurable in Single Instruction Multiple Data (SIMD) and Multiple Instruction Multiple Data (MIMD) Modes and Method of Operation, 1993. US Patent 5,212,777.

[111] GraphCore. GraphCore. https://www.graphcore.ai/, 2021. Accessed: 2021-05-16.

[112] Jordi Grau-Moya, Felix Leibfried, and Peter Vrancx. Soft q-learning with mutual-information regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HyEtjoCqFX.

[113] Alex Graves. Sequence Transduction with Recurrent Neural Networks. *arXiv preprint arXiv:1211.3711*, 2012.

[114] Arthur Gretton, Karsten M. Borgwardt, Malte J. Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *J. Mach. Learn. Res.*, 13:723–773, March 2012. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=2188385.2188410.

[115] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, et al. Bootstrap your own latent: A new approach to self-supervised learning. *arXiv preprint arXiv:2006.07733*, 2020.

[116] Caglar Gulcehre, Ziyu Wang, Alexander Novikov, Tom Le Paine, Sergio Gómez Colmenarejo, Konrad Zolna, Rishabh Agarwal, Josh Merel, Daniel Mankowitz, Cosmin Paduraru, et al. Rl unplugged: Benchmarks for offline reinforcement learning. 2020.

[117] Suriya Gunasekar, Blake E Woodworth, Srinadh Bhojanapalli, Behnam Neyshabur, and Nati Srebro. Implicit regularization in matrix factorization. In *Advances in Neural Information Processing Systems*, pages 6151–6159, 2017.

[118] Yijie Guo, Shengyu Feng, Nicolas Le Roux, Ed Chi, Honglak Lee, and Minmin Chen. Batch reinforcement learning through continuation method. In *International Conference on Learning Representations*, 2020.

[119] Abhishek Gupta, Vikash Kumar, Corey Lynch, Sergey Levine, and Karol Hausman. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. *arXiv preprint arXiv:1910.11956*, 2019.

[120] Abhishek Gupta, Aldo Pacchiano, Yuexiang Zhai, Sham M Kakade, and Sergey Levine. Unpacking reward shaping: Understanding the benefits of reward engineering on sample complexity. *arXiv preprint arXiv:2210.09579*, 2022.

[121] Suyog Gupta and Berkin Akin. Accelerator-Aware Neural Network Design using AutoML. *arXiv preprint arXiv:2003.02838*, 2020.

[122] T. Haarnoja, H. Tang, P. Abbeel, and S. Levine. Reinforcement learning with deep energy-based policies. In *International Conference on Machine Learning (ICML)*, 2017.

[123] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *arXiv*, 2018. URL https://arxiv.org/pdf/1801.01290.pdf.

[124] Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1352–1361. JMLR. org, 2017.

[125] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

[126] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[127] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. International conference on machine learning. In *International Conference on Machine Learning*, 2019.

[128] Assaf Hallak and Shie Mannor. Consistent on-line off-policy evaluation. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1372–1383. JMLR. org, 2017.

[129] Hado V Hasselt. Double q-learning. In *Advances in neural information processing systems*, pages 2613–2621, 2010.

[130] K He, X Chen, S Xie, Y Li, P Doll'ar, and RB Girshick. Masked autoencoders are scalable vision learners. *arXiv preprint arXiv.2111.06377*, 2021.

[131] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[132] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16000–16009, 2022.

[133] Yanzhang He, Tara N Sainath, Rohit Prabhavalkar, Ian McGraw, Raziel Alvarez, Ding Zhao, David Rybach, Anjuli Kannan, Yonghui Wu, Ruoming Pang, et al. Streaming End-to-End Speech Recognition for Mobile Devices. In *ICASSP*, 2019.

[134] Nicolas Heess, Gregory Wayne, David Silver, Timothy Lillicrap, Tom Erez, and Yuval Tassa. Learning continuous control policies by stochastic value gradients. In *Advances in Neural Information Processing Systems*, pages 2944–2952, 2015.

[135] Kartik Hegde, Po-An Tsai, Sitao Huang, Vikas Chandra, Angshuman Parashar, and Christopher W Fletcher. Mind Mappings: Enabling Efficient Algorithm-Accelerator Mapping Space Search. In *ASPLOS*, 2021.

[136] Matteo Hessel, Hubert Soyer, Lasse Espeholt, Wojciech Czarnecki, Simon Schmitt, and Hado van Hasselt. Multi-task deep reinforcement learning with popart. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3796–3803, 2019.

[137] Matteo Hessel, Hubert Soyer, Lasse Espeholt, Wojciech Czarnecki, Simon Schmitt, and Hado van Hasselt. Multi-task deep reinforcement learning with popart. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 2019.

[138] Todd Hester, Matej Vecerik, Olivier Pietquin, Marc Lanctot, Tom Schaul, Bilal Piot, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, et al. Deep q-learning from demonstrations. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[139] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. *Conference on Neural Information Processing Systems*, 2016.

[140] Andrew Howard and Suyog Gupta. Introducing the Next Generation of On-Device Vision Models: MobileNetV3 and MobileNetEdgeTPU. https://ai.googleblog.com/2019/11/introducing-next-generation-on-device.html, 2020.

[141] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for MobileNetV3. In *CVPR*, 2019.

[142] Zhiao Huang, Fangchen Liu, and Hao Su. Mapping state space using landmarks for universal goal reaching. *Advances in Neural Information Processing Systems*, 32: 1942–1952, 2019.

[143] Md Shahriar Iqbal, Jianhai Su, Lars Kotthoff, and Pooyan Jamshidi. FlexiBO: Cost-Aware Multi-Objective Optimization of Deep Neural Networks. *arXiv preprint arXiv:2001.06588*, 2020.

[144] Garud N Iyengar. Robust dynamic programming. *Mathematics of Operations Research*, 30(2):257–280, 2005.

[145] Arthur Jacot, Franck Gabriel, and Clement Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Advances in Neural Information Processing Systems 31*. 2018.

[146] Thomas Jaksch, Ronald Ortner, and Peter Auer. Near-optimal regret bounds for reinforcement learning. *Journal of Machine Learning Research*, 11(Apr):1563–1600, 2010.

[147] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. In *Advances in Neural Information Processing Systems*, pages 12498–12509, 2019.

[148] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. In *Advances in Neural Information Processing Systems*, pages 12498–12509, 2019.

[149] Natasha Jaques, Asma Ghandeharioun, Judy Hanwen Shen, Craig Ferguson, Agata Lapedriza, Noah Jones, Shixiang Gu, and Rosalind Picard. Way off-policy batch deep reinforcement learning of implicit human preferences in dialog. *arXiv preprint arXiv:1907.00456*, 2019.

[150] Natasha Jaques, Asma Ghandeharioun, Judy Hanwen Shen, Craig Ferguson, Àgata Lapedriza, Noah Jones, Shixiang Gu, and Rosalind W. Picard. Way off-policy batch deep reinforcement learning of implicit human preferences in dialog. *CoRR*, abs/1907.00456, 2019. URL http://arxiv.org/abs/1907.00456.

[151] Chi Jin, Qinghua Liu, and Sobhan Miryoosefi. Bellman eluder dimension: New rich classes of rl problems, and sample-efficient algorithms. *Advances in neural information processing systems*, 34:13406–13418, 2021.

[152] Ying Jin, Zhuoran Yang, and Zhaoran Wang. Is pessimism provably efficient for offline rl? In *International Conference on Machine Learning*, pages 5084–5096. PMLR, 2021.

[153] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-Datacenter Performance Analysis of a Tensor Processing Unit. In *ISCA*, 2017.

[154] Ryan Julian, Benjamin Swanson, Gaurav S Sukhatme, Sergey Levine, Chelsea Finn, and Karol Hausman. Never stop learning: The effectiveness of fine-tuning in robotic reinforcement learning. *arXiv preprint arXiv:2004.10190*, 2020.

[155] Leslie Pack Kaelbling. Learning to achieve goals. In *IJCAI*, pages 1094–1099. Citeseer, 1993.

[156] Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, et al. Model-based reinforcement learning for atari. *arXiv preprint arXiv:1903.00374*, 2019.

[157] Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *International Conference on Machine Learning (ICML)*, volume 2, 2002.

[158] Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*, pages 267–274. Morgan Kaufmann Publishers Inc., 2002.

[159] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *arXiv preprint arXiv:1806.10293*, 2018.

[160] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al. Scalable deep reinforcement learning for vision-based robotic manipulation. In *Conference on Robot Learning*, pages 651–673, 2018.

[161] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al. Scalable deep reinforcement learning for vision-based robotic manipulation. In *Conference on Robot Learning*, pages 651–673. PMLR, 2018.

[162] Dmitry Kalashnikov, Jacob Varley, Yevgen Chebotar, Benjamin Swanson, Rico Jonschkowski, Chelsea Finn, Sergey Levine, and Karol Hausman. Mt-opt: Continuous multi-task robotic reinforcement learning at scale. *arXiv preprint arXiv:2104.08212*, 2021.

[163] Bingyi Kang, Zequn Jie, and Jiashi Feng. Policy optimization with demonstrations. In *International conference on machine learning*, pages 2469–2478. PMLR, 2018.

[164] Sheng-Chun Kao, Geonhwa Jeong, and Tushar Krishna. ConfuciuX: Autonomous Hardware Resource Assignment for DNN Accelerators using Reinforcement Learning. In *MICRO*, 2020.

[165] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[166] Rahul Kidambi, Aravind Rajeswaran, Praneeth Netrapalli, and Thorsten Joachims. Morel: Model-based offline reinforcement learning. *arXiv preprint arXiv:2005.05951*, 2020.

[167] Taylor W Killian, Sonali Parbhoo, and Marzyeh Ghassemi. Risk sensitive dead-end identification in safety-critical offline reinforcement learning. *arXiv preprint arXiv:2301.05664*, 2023.

[168] Hyunjik Kim, Andriy Mnih, Jonathan Schwarz, Marta Garnelo, Ali Eslami, Dan Rosenbaum, Oriol Vinyals, and Yee Whye Teh. Attentive Neural Processes. In *ICLR*, 2019.

[169] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *ICLR*, 2015.

[170] David Koeplinger, Matthew Feldman, Raghu Prabhakar, Yaqi Zhang, Stefan Hadjis, Ruben Fiszel, Tian Zhao, Luigi Nardi, Ardavan Pedram, Christos Kozyrakis, et al. Spatial: A Language and Compiler for Application Accelerators. In *PLDI*, 2018.

[171] Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*, pages 1008–1014, 2000.

[172] Ksenia Konyushkova, Konrad Zolna, Yusuf Aytar, Alexander Novikov, Scott Reed, Serkan Cabi, and Nando de Freitas. Semi-supervised reward learning for offline reinforcement learning. *arXiv preprint arXiv:2012.06899*, 2020.

[173] Ilya Kostrikov. JAXRL: Implementations of Reinforcement Learning algorithms in JAX, 10 2021. URL https://github.com/ikostrikov/jaxrl.

[174] Ilya Kostrikov, Denis Yarats, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. *arXiv preprint arXiv:2004.13649*, 2020.

[175] Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. 2021.

[176] Ilya Kostrikov, Jonathan Tompson, Rob Fergus, and Ofir Nachum. Offline reinforcement learning with fisher divergence critic regularization. *arXiv preprint arXiv:2103.08050*, 2021.

[177] A. Kumar, A. Yazdanbakhsh, et al. Data-Driven Offline Optimization For Architecting Hardware Accelerators. *ICLR 2022*.

[178] Aviral Kumar and Sergey Levine. Model Inversion Networks for Model-Based Optimization. *NeurIPS*, 2020.

[179] Aviral Kumar, Justin Fu, Matthew Soh, George Tucker, and Sergey Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. In *Advances in Neural Information Processing Systems*, pages 11761–11771, 2019.

[180] Aviral Kumar, Abhishek Gupta, and Sergey Levine. Discor: Corrective feedback in reinforcement learning via distribution correction. *arXiv preprint arXiv:2003.07305*, 2020.

[181] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *arXiv preprint arXiv:2006.04779*, 2020.

[182] Aviral Kumar, Rishabh Agarwal, Dibya Ghosh, and Sergey Levine. Implicit under-parameterization inhibits data-efficient deep reinforcement learning. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=O9bnihsFfXU.

[183] Aviral Kumar, Rishabh Agarwal, Tengyu Ma, Aaron Courville, George Tucker, and Sergey Levine. DR3: Value-Based Deep Reinforcement Learning Requires Explicit Regularization. *arXiv preprint arXiv:2112.04716*, 2021.

[184] Aviral Kumar, Anikait Singh, Stephen Tian, Chelsea Finn, and Sergey Levine. A workflow for offline model-free robotic reinforcement learning. In *5th Annual Conference on Robot Learning*, 2021. URL https://openreview.net/forum?id=fy4ZBWxYbIo.

[185] Aviral Kumar, Rishabh Agarwal, Xinyang Geng, George Tucker, and Sergey Levine. Offline q-learning on diverse multi-task data both scales and generalizes. *arXiv preprint arXiv:2211.15144*, 2022.

[186] Aviral Kumar, Joey Hong, Anikait Singh, and Sergey Levine. Should i run offline reinforcement learning or behavioral cloning? In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=AP1MKT37rJ.

[187] Aviral Kumar, Anikait Singh, Frederik Ebert, Yanlai Yang, Chelsea Finn, and Sergey Levine. Pre-Training for Robots: Offline RL Enables Learning New Tasks from a Handful of Trials. *arXiv e-prints*, art. arXiv:2210.05178, October 2022. doi: 10.48550/arXiv.2210.05178.

[188] Aviral Kumar, Anikait Singh, Frederik Ebert, Yanlai Yang, Chelsea Finn, and Sergey Levine. Pre-training for robots: Offline rl enables learning new tasks from a handful of trials. *RSS*, 2023.

[189] Vitaly Kurin, Alessandro De Palma, Ilya Kostrikov, Shimon Whiteson, and M Pawan Kumar. In defense of the unitary scalarization for deep multi-task learning. *arXiv preprint arXiv:2201.04122*, 2022.

[190] Hyoukjun Kwon, Prasanth Chatarasi, Michael Pellauer, Angshuman Parashar, Vivek Sarkar, and Tushar Krishna. Understanding Reuse, Performance, and Hardware Cost of DNN Dataflow: A Data-Centric Approach. In *MICRO*, 2019.

[191] Michail G Lagoudakis and Ronald Parr. Least-squares policy iteration. *Journal of machine learning research*, 4(Dec):1107–1149, 2003.

[192] Sascha Lange, Thomas Gabel, and Martin Riedmiller. Batch reinforcement learning. In *Reinforcement learning*, pages 45–73. Springer, 2012.

[193] Romain Laroche, Paul Trichelair, and Rémi Tachet des Combes. Safe policy improvement with baseline bootstrapping. *arXiv preprint arXiv:1712.06924*, 2017.

[194] Romain Laroche, Paul Trichelair, and Remi Tachet Des Combes. Safe policy improvement with baseline bootstrapping. In *International Conference on Machine Learning*, pages 3652–3661. PMLR, 2019.

[195] Tor Lattimore and Csaba Szepesvári. *Bandit algorithms*. Cambridge University Press, 2020.

[196] Alex X. Lee, Anusha Nagabandi, Pieter Abbeel, and Sergey Levine. Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model. In *Advances in Neural Information Processing Systems*, 2020.

[197] Alex X Lee, Coline Devin, Jost Tobias Springenberg, Yuxiang Zhou, Thomas Lampe, Abbas Abdolmaleki, and Konstantinos Bousmalis. How to spend your robot time: Bridging kickstarting and offline reinforcement learning for vision-based robotic manipulation. *arXiv preprint arXiv:2205.03353*, 2022.

[198] Byung-Jun Lee, Jongmin Lee, and Kee-Eung Kim. Representation balancing offline model-based reinforcement learning. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=QpNz8r_Ri2Y.

[199] Kuang-Huei Lee, Ofir Nachum, Mengjiao Yang, Lisa Lee, Daniel Freeman, Winnie Xu, Sergio Guadarrama, Ian Fischer, Eric Jang, Henryk Michalewski, et al. Multi-game decision transformers. *arXiv preprint arXiv:2205.15241*, 2022.

[200] Seunghyun Lee, Younggyo Seo, Kimin Lee, Pieter Abbeel, and Jinwoo Shin. Offline-to-online reinforcement learning via balanced replay and pessimistic q-ensemble. In *Conference on Robot Learning*, pages 1702–1712. PMLR, 2022.

[201] Sergey Levine. Reinforcement learning and control as probabilistic inference: Tutorial and review. abs/1805.00909, 2018. URL http://arxiv.org/abs/1805.00909.

[202] Sergey Levine and Vladlen Koltun. Guided policy search. In *International Conference on Machine Learning*, pages 1–9, 2013.

[203] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

[204] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

[205] Alexander C Li, Lerrel Pinto, and Pieter Abbeel. Generalized hindsight for reinforcement learning. *arXiv preprint arXiv:2002.11708*, 2020.

[206] Bo Li, Anmol Gulati, Jiahui Yu, Tara N Sainath, Chung-Cheng Chiu, Arun Narayanan, Shuo-Yiin Chang, Ruoming Pang, Yanzhang He, James Qin, et al. A Better and Faster end-to-end Model for Streaming ASR. In *ICASSP*, 2021.

[207] Jiachen Li, Quan Vuong, Shuang Liu, Minghua Liu, Kamil Ciosek, Keith Ross, Henrik Iskov Christensen, and Hao Su. Multi-task batch reinforcement learning with metric learning. *arXiv preprint arXiv:1909.11373*, 2019.

[208] Qiyang Li, Yuexiang Zhai, Yi Ma, and Sergey Levine. Understanding the complexity gains of single-task rl with a curriculum. *arXiv preprint arXiv:2212.12809*, 2022.

[209] Qiyang Li, Aviral Kumar, Ilya Kostrikov, and Sergey Levine. Efficient deep reinforcement learning requires regulating overfitting. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=14-kr46GvP-.

[210] Yuanzhi Li, Colin Wei, and Tengyu Ma. Towards explaining the regularization effect of initial large learning rate in training neural networks. In *Advances in Neural Information Processing Systems*, pages 11674–11685, 2019.

[211] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[212] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

[213] Sen Lin, Jialin Wan, Tengyu Xu, Yingbin Liang, and Junshan Zhang. Model-based offline meta-reinforcement learning with regularization. *arXiv preprint arXiv:2202.02929*, 2022.

[214] Xingyu Lin, Harjatin Singh Baweja, and David Held. Reinforcement learning without ground-truth state. *arXiv preprint arXiv:1905.07866*, 2019.

[215] Changnian Liu, Yafei Tian, Qiang Zhang, Jie Yuan, and Binbin Xue. Adaptive Firefly Optimization Algorithm Based on Stochastic Inertia Weight. In *ISCID*, 2013.

[216] Hao Liu, Alexander Trott, Richard Socher, and Caiming Xiong. Competitive experience replay. *arXiv preprint arXiv:1902.00528*, 2019.

[217] Qiang Liu, Lihong Li, Ziyang Tang, and Dengyong Zhou. Breaking the curse of horizon: Infinite-horizon off-policy estimation. In *Advances in Neural Information Processing Systems*, pages 5356–5366, 2018.

[218] Yingdong Lu, Mark S Squillante, and Chai Wah Wu. A general family of robust stochastic operators for reinforcement learning. *arXiv preprint arXiv:1805.08122*, 2018.

[219] Yuping Luo, Huazhe Xu, and Tengyu Ma. Learning self-correctable policies and value functions from demonstrations with negative sampling. *arXiv preprint arXiv:1907.05634*, 2019.

[220] Corey Lynch and Pierre Sermanet. Grounding language in play. *arXiv preprint arXiv:2005.07648*, 2020.

[221] Jiafei Lyu, Xiaoteng Ma, Xiu Li, and Zongqing Lu. Mildly conservative q-learning for offline reinforcement learning. *arXiv preprint arXiv:2206.04745*, 2022.

[222] Yecheng Jason Ma, Shagun Sodhani, Dinesh Jayaraman, Osbert Bastani, Vikash Kumar, and Amy Zhang. Vip: Towards universal visual reward and representation via value-implicit pre-training. *arXiv preprint arXiv:2210.00030*, 2022.

[223] Marlos C. Machado, Marc G. Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *J. Artif. Int. Res.*, 61(1): 523–562, January 2018. ISSN 1076-9757.

[224] Hamid R. Maei, Csaba Szepesvári, Shalabh Bhatnagar, Doina Precup, David Silver, and Richard S. Sutton. Convergent temporal-difference learning with arbitrary smooth function approximation. In *Proceedings of the 22nd International Conference on Neural Information Processing Systems*, 2009.

[225] A Rupam Mahmood, Huizhen Yu, Martha White, and Richard S Sutton. Emphatic temporal-difference learning. *arXiv preprint arXiv:1507.01569*, 2015.

[226] Ajay Mandlekar, Fabio Ramos, Byron Boots, Silvio Savarese, Li Fei-Fei, Animesh Garg, and Dieter Fox. Iris: Implicit reinforcement without interaction at scale for learning control from offline robot manipulation data. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4414–4420. IEEE, 2020.

[227] Ajay Mandlekar, Danfei Xu, Josiah Wong, Soroush Nasiriany, Chen Wang, Rohun Kulkarni, Fei-Fei Li, Silvio Savarese, Yuke Zhu, and Roberto Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation. In *5th Annual Conference on Robot Learning*, 2021. URL https://openreview.net/forum?id=JrsfBJtDFdI.

[228] Max Sobol Mark, Ali Ghadirzadeh, Xi Chen, and Chelsea Finn. Fine-tuning offline policies with optimistic action selection. In *Deep Reinforcement Learning Workshop NeurIPS 2022*, 2022.

[229] Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nazi, et al. A Graph Placement Methodology for Fast Chip Design. *Nature*, 2021.

[230] Eric Mitchell, Rafael Rafailov, Xue Bin Peng, Sergey Levine, and Chelsea Finn. Offline meta-reinforcement learning with advantage weighting. In *International Conference on Machine Learning*, pages 7780–7791. PMLR, 2021.

[231] Akshita Mittel and Purna Sowmya Munukutla. Visual transfer between atari games using competitive reinforcement learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 0–0, 2019.

[232] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[233] Rotem Mulayoff and Tomer Michaeli. Unique properties of flat minima in deep networks. In *International Conference on Machine Learning*, pages 7108–7118. PMLR, 2020.

[234] Rémi Munos. Error bounds for approximate policy iteration. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, pages 560–567. AAAI Press, 2003.

[235] Rémi Munos. Error bounds for approximate value iteration. In *Proceedings of the National Conference on Artificial Intelligence*, 2005.

[236] Rémi Munos, Tom Stepleton, Anna Harutyunyan, and Marc Bellemare. Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 1054–1062, 2016.

[237] Ofir Nachum, Yinlam Chow, Bo Dai, and Lihong Li. Dualdice: Behavior-agnostic estimation of discounted stationary distribution corrections. In *Advances in Neural Information Processing Systems*, pages 2315–2325, 2019.

[238] Ofir Nachum, Bo Dai, Ilya Kostrikov, Yinlam Chow, Lihong Li, and Dale Schuurmans. Algaedice: Policy gradient from arbitrary experience. *arXiv preprint arXiv:1912.02074*, 2019.

[239] Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7559–7566. IEEE, 2018.

[240] Ashvin Nair, Murtaza Dalal, Abhishek Gupta, and Sergey Levine. Accelerating online reinforcement learning with offline datasets. *CoRR*, abs/2006.09359, 2020. URL https://arxiv.org/abs/2006.09359.

[241] Ashvin Nair, Murtaza Dalal, Abhishek Gupta, and Sergey Levine. Accelerating online reinforcement learning with offline datasets. *arXiv preprint arXiv:2006.09359*, 2020.

[242] Ashvin V Nair, Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine. Visual reinforcement learning with imagined goals. *Advances in neural information processing systems*, 31, 2018.

[243] Suraj Nair, Aravind Rajeswaran, Vikash Kumar, Chelsea Finn, and Abhinav Gupta. R3m: A universal visual representation for robot manipulation. *arXiv preprint arXiv:2203.12601*, 2022.

[244] Mitsuhiko Nakamoto, Yuexiang Zhai, Anikait Singh, Max Sobol Mark, Yi Ma, Chelsea Finn, Aviral Kumar, and Sergey Levine. Cal-QL: Calibrated offline rl pre-training for efficient online fine-tuning. *arXiv preprint arXiv:2303.05479*, 2023.

[245] Hongseok Namkoong and John C Duchi. Variance-based regularization with convex objectives. In *Advances in neural information processing systems*, pages 2971–2980, 2017.

[246] Luigi Nardi, David Koeplinger, and Kunle Olukotun. Practical Design Space Exploration. In *MASCOTS*, 2019.

[247] Andrew Y. Ng and Stuart J. Russell. Algorithms for inverse reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, 2000.

[248] Arnab Nilim and Laurent El Ghaoui. Robustness in markov decision problems with uncertain transition matrices. In *Advances in neural information processing systems*, pages 839–846, 2004.

[249] Nvidia. NVDLA deep learning accelerator. http://nvdla.org, 2021. Accessed: 2021-10-01.

[250] Nvidia. Nvidia. https://www.nvidia.com/en-us/, 2021. Accessed: 2021-05-16.

[251] Brendan O'Donoghue. Variational bayesian reinforcement learning with regret bounds. *arXiv preprint arXiv:1807.09647*, 2018.

[252] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[253] Ian Osband and Benjamin Van Roy. Why is posterior sampling better than optimism for reinforcement learning? In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2701–2710. JMLR. org, 2017.

[254] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. In *Advances in neural information processing systems*, pages 4026–4034, 2016.

[255] Angshuman Parashar, Priyanka Raina, Yakun Sophia Shao, Yu-Hsin Chen, Victor A Ying, Anurag Mukkara, Rangharajan Venkatesan, Brucek Khailany, Stephen W Keckler, and Joel Emer. Timeloop: A Systematic Approach to DNN Accelerator Evaluation. In *ISPASS*. IEEE, 2019.

[256] Emilio Parisotto, Jimmy Lei Ba, and Ruslan Salakhutdinov. Actor-mimic: Deep multitask and transfer reinforcement learning. *arXiv preprint arXiv:1511.06342*, 2015.

[257] Paavo Parmas, Carl Edward Rasmussen, Jan Peters, and Kenji Doya. Pipps: Flexible model-based policy search robust to the curse of chaos. *arXiv preprint arXiv:1902.01240*, 2019.

[258] Maryam Parsa, John P Mitchell, Catherine D Schuman, Robert M Patton, Thomas E Potok, and Kaushik Roy. Bayesian Multi-objective Hyperparameter Optimization for Accurate, Fast, and Efficient Neural Network Accelerator Design. *Frontiers in Neuroscience*, 14:667, 2020.

[259] Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *arXiv preprint arXiv:1910.00177*, 2019.

[260] Leonid Peshkin and Christian R Shelton. Learning from scarce experience. *arXiv preprint cs/0204043*, 2002.

[261] Marek Petrik, Mohammad Ghavamzadeh, and Yinlam Chow. Safe policy improvement by minimizing robust baseline regret. In *Advances in Neural Information Processing Systems*, pages 2298–2306, 2016.

[262] Matthias Plappert, Marcin Andrychowicz, Alex Ray, Bob McGrew, Bowen Baker, Glenn Powell, Jonas Schneider, Josh Tobin, Maciek Chociej, Peter Welinder, Vikash Kumar, and Wojciech Zaremba. Multi-goal reinforcement learning: Challenging robotics environments and request for research, 2018.

[263] Tobias Pohlen, Bilal Piot, Todd Hester, Mohammad Gheshlaghi Azar, Dan Horgan, David Budden, Gabriel Barth-Maron, Hado Van Hasselt, John Quan, Mel Večerík, et al. Observe and look further: Achieving consistent performance on atari. *arXiv preprint arXiv:1805.11593*, 2018.

[264] Dean A Pomerleau. Alvinn: an autonomous land vehicle in a neural network. In *Proceedings of the 1st International Conference on Neural Information Processing Systems*, pages 305–313, 1988.

[265] Vitchyr Pong, Shixiang Gu, Murtaza Dalal, and Sergey Levine. Temporal difference models: Model-free deep rl for model-based control. *arXiv preprint arXiv:1802.09081*, 2018.

[266] Vitchyr H Pong, Ashvin Nair, Laura Smith, Catherine Huang, and Sergey Levine. Offline meta-reinforcement learning with online self-supervision. *arXiv preprint arXiv:2107.03974*, 2021.

[267] Prakruthi Prabhakar, Yiping Yuan, Guangyu Yang, Wensheng Sun, and Ajith Muralidharan. Multi-objective optimization of notifications using offline reinforcement learning. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 3752–3760, 2022.

[268] Doina Precup. Eligibility traces for off-policy policy evaluation. *Computer Science Department Faculty Publication Series*, page 80, 2000.

[269] Doina Precup, Richard S Sutton, and Sanjoy Dasgupta. Off-policy temporal-difference learning with function approximation. In *ICML*, pages 417–424, 2001.

[270] Doina Precup, Richard S. Sutton, and Sanjoy Dasgupta. Off-policy temporal-difference learning with function approximation. In *International Conference on Machine Learning (ICML)*, 2001.

[271] Ilija Radosavovic, Tete Xiao, Stephen James, Pieter Abbeel, Jitendra Malik, and Trevor Darrell. Real-world robot learning with masked visual pre-training. *arXiv preprint arXiv:2210.03109*, 2022.

[272] Rafael Rafailov, Tianhe Yu, A. Rajeswaran, and Chelsea Finn. Offline reinforcement learning from images with latent space models. *Learning for Decision Making and Control (L4DC)*, 2021.

[273] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. *arXiv preprint arXiv:1709.10087*, 2017.

[274] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. In *Robotics: Science and Systems*, 2018.

[275] Brandon Reagen, José Miguel Hernández-Lobato, Robert Adolf, Michael Gelbart, Paul Whatmough, Gu-Yeon Wei, and David Brooks. A Case for Efficient Accelerator Design Space Exploration via Bayesian Optimization. In *ISLPED*, 2017.

[276] Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, et al. A generalist agent. *arXiv preprint arXiv:2205.06175*, 2022.

[277] Martin Riedmiller. Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method. In *European Conference on Machine Learning*, pages 317–328. Springer, 2005.

[278] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *MICCAI*. Springer, 2015.

[279] Stephane Ross and Drew Bagnell. Agnostic system identification for model-based reinforcement learning. In *ICML*, 2012.

[280] Andrei A Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation. *arXiv preprint arXiv:1511.06295*, 2015.

[281] Tara N Sainath, Yanzhang He, Bo Li, Arun Narayanan, Ruoming Pang, Antoine Bruguier, Shuo-yiin Chang, Wei Li, Raziel Alvarez, Zhifeng Chen, et al. A Streaming On-Device End-to-End Model Surpassing Server-Side Conventional Model Quality and Latency. In *ICASSP*, 2020.

[282] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In *CVPR*, 2018.

[283] Stefan Schaal. Learning from demonstration. *Advances in neural information processing systems*, 9, 1996.

[284] Stefan Schaal. Is imitation learning the route to humanoid robots?, 1999.

[285] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *International conference on machine learning*, pages 1312–1320. PMLR, 2015.

[286] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *International Conference on Learning Representations (ICLR)*, 2015.

[287] Tom Schaul, Diana Borsa, Joseph Modayil, and Razvan Pascanu. Ray interference: a source of plateaus in deep reinforcement learning. *CoRR*, abs/1904.11455, 2019. URL http://arxiv.org/abs/1904.11455.

[288] Bruno Scherrer, Mohammad Ghavamzadeh, Victor Gabillon, Boris Lesner, and Matthieu Geist. Approximate modified policy iteration and its application to the game of tetris. *Journal of Machine Learning Research*, 16:1629–1676, 2015. URL http://jmlr.org/papers/v16/scherrer15a.html.

[289] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel. Trust region policy optimization. In *International Conference on Machine Learning (ICML)*, 2015.

[290] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015.

[291] Nur Muhammad Mahi Shafiullah, Zichen Jeff Cui, Ariuntuya Altanzaya, and Lerrel Pinto. Behavior transformers: Cloning k modes with one stone. *arXiv preprint arXiv:2206.11251*, 2022.

[292] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P. Adams, and Nando de Freitas. Taking the Human Out of the Loop: A Review of Bayesian Optimization. *Proceedings of the IEEE*, 2016.

[293] Pratyusha Sharma, Lekha Mohan, Lerrel Pinto, and Abhinav Gupta. Multiple interactions made easy (mime): Large scale demonstrations data for imitation. In *Conference on robot learning*, pages 906–915. PMLR, 2018.

[294] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer. *arXiv preprint arXiv:1701.06538*, 2017.

[295] Zhan Shi, Chirag Sakhuja, Milad Hashemi, Kevin Swersky, and Calvin Lin. Learned Hardware/Software Co-Design of Neural Accelerators. *arXiv preprint arXiv:2010.02075*, 2020.

[296] Noah Y Siegel, Jost Tobias Springenberg, Felix Berkenkamp, Abbas Abdolmaleki, Michael Neunert, Thomas Lampe, Roland Hafner, and Martin Riedmiller. Keep doing what worked: Behavioral modelling priors for offline reinforcement learning. *arXiv preprint arXiv:2002.08396*, 2020.

[297] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[298] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

[299] Thiago D Simão, Romain Laroche, and Rémi Tachet des Combes. Safe policy improvement with an estimated baseline policy. *arXiv preprint arXiv:1909.05236*, 2019.

[300] Anikait Singh, Aviral Kumar, Quan Vuong, Yevgen Chebotar, and Sergey Levine. Offline rl with realistic datasets: Heteroskedasticity and support constraints. *arXiv preprint arXiv:2211.01052*, 2022.

[301] Avi Singh, Larry Yang, Kristian Hartikainen, Chelsea Finn, and Sergey Levine. End-to-end robotic reinforcement learning without reward engineering. *arXiv preprint arXiv:1904.07854*, 2019.

[302] Avi Singh, Huihan Liu, Gaoyue Zhou, Albert Yu, Nicholas Rhinehart, and Sergey Levine. Parrot: Data-driven behavioral priors for reinforcement learning. *arXiv preprint arXiv:2011.10024*, 2020.

[303] Avi Singh, Albert Yu, Jonathan Yang, Jesse Zhang, Aviral Kumar, and Sergey Levine. Cog: Connecting new skills to past experience with offline reinforcement learning. *arXiv preprint arXiv:2010.14500*, 2020.

[304] Jasper Snoek, Hugo Larochelle, and Ryan P. Adams. Practical Bayesian Optimization of Machine Learning Algorithms. In *NIPS*, 2012.

[305] Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Mostofa Patwary, Mr Prabhat, and Ryan Adams. Scalable Bayesian Optimization Using Deep Neural Networks. In *ICML*, 2015.

[306] Shagun Sodhani, Amy Zhang, and Joelle Pineau. Multi-task reinforcement learning with context-based representations. In *International Conference on Machine Learning*, pages 9767–9779. PMLR, 2021.

[307] H Francis Song, Abbas Abdolmaleki, Jost Tobias Springenberg, Aidan Clark, Hubert Soyer, Jack W Rae, Seb Noury, Arun Ahuja, Siqi Liu, Dhruva Tirumala, et al. V-mpo: On-policy maximum a posteriori policy optimization for discrete and continuous control. *arXiv preprint arXiv:1909.12238*, 2019.

[308] Yuda Song, Yifei Zhou, Ayush Sekhari, Drew Bagnell, Akshay Krishnamurthy, and Wen Sun. Hybrid RL: Using both offline and online data can make RL efficient. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=yyBis80iUuU.

[309] Hao Sun, Zhizhong Li, Xiaotong Liu, Dahua Lin, and Bolei Zhou. Policy continuation with hindsight inverse dynamics. *arXiv preprint arXiv:1910.14055*, 2019.

[310] Wen Sun, Nan Jiang, Akshay Krishnamurthy, Alekh Agarwal, and John Langford. Model-based rl in contextual decision processes: Pac bounds and exponential improvements over model-free approaches. In *Conference on learning theory*, pages 2898–2933. PMLR, 2019.

[311] Yanan Sun, Bing Xue, Mengjie Zhang, Gary G Yen, and Jiancheng Lv. Automatically Designing CNN Architectures Using the Genetic Algorithm for Image Classification. *IEEE Transactions on Cybernetics*, 2020.

[312] R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998. ISBN 0262193981.

[313] Richard S Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM Sigart Bulletin*, 2(4):160–163, 1991.

[314] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. Second edition, 2018.

[315] Richard S. Sutton, A. Rupam Mahmood, and Martha White. An emphatic approach to the problem of off-policy temporal-difference learning. *J. Mach. Learn. Res.*, 17(1): 2603–2631, January 2016. ISSN 1532-4435.

[316] Aviv Tamar, Shie Mannor, and Huan Xu. Scaling up robust mdps using function approximation. In *International Conference on Machine Learning*, pages 181–189, 2014.

[317] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.

[318] Yuval Tassa, Tom Erez, and Emanuel Todorov. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4906–4913. IEEE, 2012.

[319] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

[320] Yee Whye Teh, Victor Bapst, Wojciech Marian Czarnecki, John Quan, James Kirkpatrick, Raia Hadsell, Nicolas Heess, and Razvan Pascanu. Distral: Robust multitask reinforcement learning. *arXiv preprint arXiv:1707.04175*, 2017.

[321] Gerald Tesauro. TD-Gammon, a self-teaching backgammon program, achieves master-level play. *Neural computation*, 6(2):215–219, 1994.

[322] Philip Thomas, Georgios Theocharous, and Mohammad Ghavamzadeh. High confidence policy improvement. In *International Conference on Machine Learning*, pages 2380–2388, 2015.

[323] Yuandong Tian, Lantao Yu, Xinlei Chen, and Surya Ganguli. Understanding self-supervised learning with dual deep networks. *arXiv preprint arXiv:2010.00578*, 2020.

[324] Yuandong Tian, Xinlei Chen, and Surya Ganguli. Understanding self-supervised learning dynamics without contrastive pairs. *arXiv preprint arXiv:2102.06810*, 2021.

[325] Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *IROS*, pages 5026–5033, 2012.

[326] Brandon Trabucco, Aviral Kumar, Xinyang Geng, and Sergey Levine. Conservative Objective Models for Effective Offline Model-Based Optimization. In *ICML*, 2021.

[327] Brandon Trabucco, Aviral Kumar, Xinyang Geng, and Sergey Levine. Design-Bench: Benchmarks for Data-Driven Offline Model-Based Optimization, 2021. URL https://openreview.net/forum?id=cQzf26aA3vM.

[328] Hado van Hasselt, Yotam Doron, Florian Strub, Matteo Hessel, Nicolas Sonnerat, and Joseph Modayil. Deep reinforcement learning and the deadly triad. *ArXiv*, abs/1812.02648, 2018.

[329] Hado Van Hasselt, Yotam Doron, Florian Strub, Matteo Hessel, Nicolas Sonnerat, and Joseph Modayil. Deep reinforcement learning and the deadly triad. *arXiv preprint arXiv:1812.02648*, 2018.

[330] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[331] Mel Vecerik, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Heess, Thomas Rothörl, Thomas Lampe, and Martin Riedmiller. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint arXiv:1707.08817*, 2017.

[332] Rangharajan Venkatesan, Yakun Sophia Shao, Miaorong Wang, Jason Clemons, Steve Dai, Matthew Fojtik, Ben Keller, Alicia Klinefelter, Nathaniel Pinckney, Priyanka Raina, et al. MAGNet: A Modular Accelerator Generator for Neural Networks. In *ICCAD*, 2019.

[333] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[334] Andrew Wagenmaker and Aldo Pacchiano. Leveraging offline data in online reinforcement learning. *arXiv preprint arXiv:2211.04974*, 2022.

[335] Ruosong Wang, Dean Foster, and Sham M. Kakade. What are the statistical limits of offline {rl} with linear function approximation? In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=30EvkP2aQLD.

[336] Ruosong Wang, Yifan Wu, Ruslan Salakhutdinov, and Sham M Kakade. Instabilities of offline rl with pre-trained neural representation. *arXiv preprint arXiv:2103.04947*, 2021.

[337] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4): 279–292, 1992.

[338] Colin Wei, Jason Lee, Qiang Liu, and Tengyu Ma. Regularization matters: Generalization and optimization of neural nets vs their induced kernel. 2019.

[339] Aaron Wilson, Alan Fern, Soumya Ray, and Prasad Tadepalli. Multi-task reinforcement learning: a hierarchical bayesian approach. In *Proceedings of the 24th international conference on Machine learning*, pages 1015–1022, 2007.

[340] William Wong, Praneet Dutta, Octavian Voicu, Yuri Chervonyi, Cosmin Paduraru, and Jerry Luo. Optimizing industrial hvac systems with hierarchical reinforcement learning. *arXiv preprint arXiv:2209.08112*, 2022.

[341] Blake Woodworth, Suriya Gunasekar, Jason D Lee, Edward Moroshko, Pedro Savarese, Itay Golan, Daniel Soudry, and Nathan Srebro. Kernel and rich regimes in overparametrized models. *arXiv preprint arXiv:2002.09277*, 2020.

[342] Jialong Wu, Haixu Wu, Zihan Qiu, Jianmin Wang, and Mingsheng Long. Supported policy optimization for offline reinforcement learning. *arXiv preprint arXiv:2202.06239*, 2022.

[343] Yifan Wu, George Tucker, and Ofir Nachum. Behavior regularized offline reinforcement learning. *arXiv preprint arXiv:1911.11361*, 2019.

[344] Yuxin Wu and Kaiming He. Group normalization. In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018.

[345] Chenjun Xiao, Han Wang, Yangchen Pan, Adam White, and Martha White. The in-sample softmax for offline reinforcement learning. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=u-RuvyDYqCM.

[346] Tete Xiao, Ilija Radosavovic, Trevor Darrell, and Jitendra Malik. Masked visual pre-training for motor control. *arXiv preprint arXiv:2203.06173*.

[347] Annie Xie, Avi Singh, Sergey Levine, and Chelsea Finn. Few-shot goal inference for visuomotor learning and planning. In *Conference on Robot Learning*, pages 40–52. PMLR, 2018.

[348] Annie Xie, Frederik Ebert, Sergey Levine, and Chelsea Finn. Improvisation through physical understanding: Using novel objects as tools with visual foresight. *Robotics: Science and Systems (RSS)*, 2019.

[349] Tengyang Xie and Nan Jiang. Q* approximation schemes for batch reinforcement learning: A eoretical comparison. 2020.

[350] Tengyang Xie, Ching-An Cheng, Nan Jiang, Paul Mineiro, and Alekh Agarwal. Bellman-consistent pessimism for offline reinforcement learning. *Advances in neural information processing systems*, 34, 2021.

[351] Tengyang Xie, Nan Jiang, Huan Wang, Caiming Xiong, and Yu Bai. Policy finetuning: Bridging sample-efficient offline and online reinforcement learning. *Advances in neural information processing systems*, 34:27395–27407, 2021.

[352] Pengfei Xu, Xiaofan Zhang, Cong Hao, Yang Zhao, Yongan Zhang, Yue Wang, Chaojian Li, Zetong Guan, Deming Chen, and Yingyan Lin. AutoDNNchip: An Automated DNN Chip Predictor and Builder for both FPGAs and ASICs. In *FPGA*, 2020.

[353] Zhiyuan Xu, Kun Wu, Zhengping Che, Jian Tang, and Jieping Ye. Knowledge transfer in multi-task deep reinforcement learning for continuous control. *arXiv preprint arXiv:2010.07494*, 2020.

[354] Mengjiao Yang and Ofir Nachum. Representation matters: Offline pretraining for sequential decision making. *arXiv preprint arXiv:2102.05815*, 2021.

[355] Mengjiao Yang, Sergey Levine, and Ofir Nachum. Trail: Near-optimal imitation learning with suboptimal data. *arXiv preprint arXiv:2110.14770*, 2021.

[356] Rui Yang, Jiafei Lyu, Yu Yang, Jiangpeng Ya, Feng Luo, Dijun Luo, Lanqing Li, and Xiu Li. Bias-reduced multi-step hindsight experience replay. *arXiv preprint arXiv:2102.12962*, 2021.

[357] Ruihan Yang, Huazhe Xu, Yi Wu, and Xiaolong Wang. Multi-task reinforcement learning with soft modularization. *arXiv preprint arXiv:2003.13661*, 2020.

[358] Xin-She Yang. *Nature-Inspired Metaheuristic Algorithms*. Luniver press, 2010.

[359] Xin-She Yang and Suash Deb. Eagle Strategy Using Lévy Walk and Firefly Algorithms for Stochastic Optimization. In *NICSO*. Springer, 2010.

[360] Amir Yazdanbakhsh, Christof Angermueller, Berkin Akin, Yanqi Zhou, Albin Jones, Milad Hashemi, Kevin Swersky, Satrajit Chatterjee, Ravi Narayanaswami, and James Laudon. Apollo: Transferable Architecture Exploration. *arXiv preprint arXiv:2102.01723*, 2021.

[361] Amir Yazdanbakhsh, Kiran Seshadri, Berkin Akin, James Laudon, and Ravi Narayanaswami. An Evaluation of Edge TPU Accelerators for Convolutional Neural Networks. *arXiv preprint arXiv:2102.10423*, 2021.

[362] Deheng Ye, Guibin Chen, Wen Zhang, Sheng Chen, Bo Yuan, Bo Liu, Jia Chen, Zhao Liu, Fuhao Qiu, Hongsheng Yu, Yinyuting Yin, Bei Shi, Liang Wang, Tengfei Shi, Qiang Fu, Wei Yang, Lanxiao Huang, and Wei Liu. Towards playing full moba games with deep reinforcement learning. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 621–632. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/06d5ae105ea1bea4d800bc96491876e9-Paper.pdf.

[363] Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning. *arXiv preprint arXiv:2001.06782*, 2020.

[364] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning*, pages 1094–1100. PMLR, 2020.

[365] Tianhe Yu, Garrett Thomas, Lantao Yu, Stefano Ermon, James Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. Mopo: Model-based offline policy optimization. *arXiv preprint arXiv:2005.13239*, 2020.

[366] Tianhe Yu, Aviral Kumar, Yevgen Chebotar, Karol Hausman, Sergey Levine, and Chelsea Finn. Conservative data sharing for multi-task offline reinforcement learning. *Advances in Neural Information Processing Systems*, 34, 2021.

[367] Tianhe Yu, Aviral Kumar, Rafael Rafailov, Aravind Rajeswaran, Sergey Levine, and Chelsea Finn. Combo: Conservative offline model-based policy optimization. *arXiv preprint arXiv:2102.08363*, 2021.

[368] Tianhe Yu, Aviral Kumar, Yevgen Chebotar, Karol Hausman, Chelsea Finn, and Sergey Levine. How to leverage unlabeled data in offline reinforcement learning. *arXiv preprint arXiv:2202.01741*, 2022.

[369] Andrea Zanette. Exponential lower bounds for batch reinforcement learning: Batch rl can be exponentially harder than online rl. *arXiv preprint arXiv:2012.08005*, 2020.

[370] Andrea Zanette, Martin J Wainwright, and Emma Brunskill. Provable benefits of actor-critic methods for offline reinforcement learning. *Advances in neural information processing systems*, 34:13626–13640, 2021.

[371] Yuexiang Zhai, Christina Baek, Zhengyuan Zhou, Jiantao Jiao, and Yi Ma. Computational benefits of intermediate rewards for goal-reaching policy learning. *Journal of Artificial Intelligence Research*, 73:847–896, 2022.

[372] Chiyuan Zhang, Samy Bengio, and Yoram Singer. Are all layers created equal? *arXiv preprint arXiv:1902.01996*, 2019.

[373] Ruiyi Zhang, Bo Dai, Lihong Li, and Dale Schuurmans. Gendice: Generalized offline estimation of stationary values. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=HkxlcnVFwB.

[374] Qinqing Zheng, Amy Zhang, and Aditya Grover. Online decision transformer. In *International Conference on Machine Learning*, pages 27042–27059. PMLR, 2022.

[375] Henry Zhu, Abhishek Gupta, Aravind Rajeswaran, Sergey Levine, and Vikash Kumar. Dexterous manipulation with deep reinforcement learning: Efficient, general, and low-cost. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 3651–3657. IEEE, 2019.

[376] Yuke Zhu, Ziyu Wang, Josh Merel, Andrei Rusu, Tom Erez, Serkan Cabi, Saran Tunyasuvunakool, János Kramár, Raia Hadsell, Nando de Freitas, et al. Reinforcement and imitation learning for diverse visuomotor skills. *arXiv preprint arXiv:1802.09564*, 2018.

[377] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, and Anind K Dey. Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.

Part V

APPENDICES

A

APPENDIX: CHALLENGES IN OFFLINE REINFORCEMENT LEARNING

A.1 BENCHMARK TABULAR DOMAINS

We evaluate on a benchmark of 8 tabular domains, selected for qualitative differences.

Gridworlds. The Gridworld task is an NxN grid with randomly placed walls. The reward is proportional to Manhattan distance to a goal state (1 at the goal, 0 at the initial position), and there is a 5% chance the agent travels in a different direction than commanded. We vary two parameters: the size (16×16 and 64×64), and the state representations. We use a "one-hot" representation, an (X, Y) coordinate tuple (represented as two one-hot vectors), and a "random" representation, a vector drawn from $\mathcal{N}(0,1)^N$, where N is the width or height of the Gridworld. Random observations significantly increase the difficulty, as significant state aliasing occurs.

Cliffwalk: Cliffwalk is a toy example from Schaul et al. [286]. It consists of a sequence of states, where each state has two allowed actions: advance to the next state or return to the initial state. A reward of 1.0 is obtained when the agent reaches the final state. Observations consist of vectors drawn from $\mathcal{N}(0,1)^{16}$.

InvertedPendulum and MountainCar: InvertedPendulum and MountainCar are discretized versions of continuous control tasks found in OpenAI gym [262], and are based on problems from classical RL literature. In the InvertedPendulum task, an agent must swing up an pendulum and hold it in its upright position. The state consists of the angle and angular velocity of the pendulum. Maximum reward is given when the pendulum is upright. The observation consists of the sin and cos of the pendulum angle, and the angular velocity. In MountainCar, the agent must push a vehicle up a hill, but the hill is steep that the agent must gather momentum by swinging back and forth within a valley to reach the top. The state consists of the position and velocity of the vehicle.

SparseGraph: The SparseGraph environment is a 256-state graph with randomly drawn edges. Each state has two edges, each corresponding to an action. One state is chosen as the goal state, where the agent receives a reward of one.

B

APPENDIX: RESTRICTING ACTION SELECTION FOR POLICY LEARNING

B.1 DISTRIBUTION-CONSTRAINED BACKUP OPERATOR

In this section, we analyze properties of the constrained Bellman backup operator, defined as:

$$\mathcal{B}^{\Pi}Q(\boldsymbol{s},\boldsymbol{a}) \stackrel{\text{def}}{=} \mathbb{E}\left[r(\boldsymbol{s},\boldsymbol{a}) + \gamma \max_{\pi\in\Pi} \mathbb{E}_{T(\boldsymbol{s}'|\boldsymbol{s},\boldsymbol{a})}\left[V(s')\right]\right]$$

where

$$V(\boldsymbol{s}) \stackrel{\text{def}}{=} \max_{\pi\in\Pi} \mathbb{E}_{\pi}[Q(\boldsymbol{s},\boldsymbol{a})].$$

Such an operator can be reduced to a standard Bellman backup in a modified MDP. We can construct an MDP M' from the original MDP M as follows:

- The state space, discount, and initial state distributions remain unchanged from M.
- We define a new action set $\mathcal{A}' = \Pi$ to be the choice of policy π to execute.
- We define the new transition distribution p' as taking one step under the chosen policy π to execute and one step under the original dynamics T: $T'(\boldsymbol{s}'|\boldsymbol{s},\pi) = E_{\pi}[T(s'|s,a)]$.
- Q-values in this new MDP, $Q^{\Pi}(\boldsymbol{s},\pi)$ would, in words, correspond to executing policy π for one step and executing the policy which maximizes the future discounted value function in the original MDP M thereafter.

Under this redefinition, the Bellman operator $\mathcal{B}^{\Pi}$ is mathematically the same operation as the Bellman operator under M'. Thus, standard results from MDP theory carry over - i.e. the existence of a fixed point and convergence of repeated application of $\mathcal{B}^{\Pi}$ to said fixed point.

In this section, we provide proofs for Theorem 4.3.4 and Theorem 4.3.5.

Theorem B.2.1. *Suppose we run approximate distribution-constrained value iteration with a set constrained backup $\mathcal{B}^\Pi$. Assume that $\delta(\boldsymbol{s}, \boldsymbol{a}) \geq \max_k |Q_k(\boldsymbol{s}, \boldsymbol{a}) - \mathcal{B}^\Pi Q_{k-1}(\boldsymbol{s}, \boldsymbol{a})|$ bounds the Bellman error. Then,*

$$\lim_{k\to\infty} \mathbb{E}_{\rho_0}[|V_k(\boldsymbol{s}) - V^*(\boldsymbol{s})|] \leq \frac{\gamma}{(1-\gamma)^2}\left[C(\Pi)\mathbb{E}_\mu[\max_{\pi\in\Pi} \mathbb{E}_\pi[\delta(\boldsymbol{s}, \boldsymbol{a})]] + \frac{1-\gamma}{\gamma}\alpha(\Pi)\right]$$

Proof. We first begin by introducing V^Π, the fixed point of $\mathcal{B}^\Pi$. By the triangle inequality, we have:

$$\begin{aligned}\mathbb{E}_{\rho_0}[|V_k(\boldsymbol{s}) - V^*(\boldsymbol{s})|] &= \mathbb{E}_{\rho_0}[|V_k(\boldsymbol{s}) - V^\Pi(\boldsymbol{s}) + V^\Pi(\boldsymbol{s}) - V^*(\boldsymbol{s})|] \\ &\leq \underbrace{\mathbb{E}_{\rho_0}[|V_k(\boldsymbol{s}) - V^\Pi(\boldsymbol{s})|]}_{L_1} + \underbrace{\mathbb{E}_{\rho_0}[|V^\Pi(\boldsymbol{s}) - V^*(\boldsymbol{s})|]}_{L_2}\end{aligned}$$

First, we note that $\max_\pi \mathbb{E}_\pi[\delta(\boldsymbol{s}, \boldsymbol{a})]$ provides an upper bound on the value error:

$$\begin{aligned}|V_k(\boldsymbol{s}) - \mathcal{B}^\Pi V_{k-1}(\boldsymbol{s})| &= |\max_\pi \mathbb{E}_\pi[Q_k(\boldsymbol{s}, \boldsymbol{a})] - \max_\pi \mathbb{E}_\pi[\mathcal{B}^\pi Q_{k-1}(\boldsymbol{s}, \boldsymbol{a})]| \\ &\leq \max_\pi \mathbb{E}_\pi[|Q_k(\boldsymbol{s}, \boldsymbol{a}) - \mathcal{B}^\pi Q_{k-1}(\boldsymbol{s}, \boldsymbol{a})|] \\ &\leq \max_\pi \mathbb{E}_\pi[\delta(\boldsymbol{s}, \boldsymbol{a})]\end{aligned}$$

We can bound L_1 with

$$L_1 \leq \frac{2\gamma}{(1-\gamma)^2}[C(\Pi)]\mathbb{E}_\mu[\max_{\pi\in\Pi} \mathbb{E}_\pi[\delta(\boldsymbol{s}, \boldsymbol{a})]]$$

by direct modification of the proof of Theorem 3 of Farahmand et al. [79] or Theorem 1 of Munos [235] with $k = 1$ ($p = 1$), but replacing V^* with V^Π and $\mathcal{B}$ with $\mathcal{B}^\Pi$, as $\mathcal{B}^\Pi$ is a contraction and V^Π is its fixed point. An alternative proof involves viewing $\mathcal{B}^\Pi$ as a backup under a modified MDP (see Appendix B.1), and directly apply Theorem 1 of Munos [235] under this modified MDP. A similar bound also holds true for value iteration with the $\mathcal{B}^\Pi$ operator which can be analysed on similar lines as the above proof and Munos [235].

To bound L_2, we provide a simple ℓ_∞-norm bound, although we could in principle apply techniques used to bound L_1 to get a tighter distribution-based bound.

$$\begin{aligned}V^\Pi - V^* &= \mathcal{B}^\Pi V^\Pi - \mathcal{B}V^* \\ &\leq \mathcal{B}^\Pi V^\Pi - \mathcal{B}^\Pi V^* + \mathcal{B}^\Pi V^\Pi - \mathcal{B}V^* \\ &\leq \gamma V^\Pi - V^* + \alpha(\Pi)\end{aligned}$$

Thus, we have $V^{\Pi} - V^{*} \leq \frac{\alpha}{1-\gamma}$. Because the maximum is greater than the expectation, $L_2 = \mathbb{E}_{\rho_0,\pi}[|V^{\Pi}(\boldsymbol{s}) - V^{*}(\boldsymbol{s})|] \leq V^{\Pi} - V^{*}$.

Adding L_1 and L_2 completes the proof. □

Theorem B.2.2. *Assume the data distribution μ is generated by a behavior policy β, such that $\mu(\boldsymbol{s}, \boldsymbol{a}) = \mu_\beta(\boldsymbol{s}, \boldsymbol{a})$. Let $\mu(\boldsymbol{s})$ be the marginal state distribution under the data distribution. Let us define $\Pi_\epsilon = \{\pi \mid \pi(\boldsymbol{a}|\boldsymbol{s}) = 0$ whenever $\beta(\boldsymbol{a}|\boldsymbol{s}) < \epsilon\}$. Then, there exists a concentrability coefficient $C(\Pi_\epsilon)$ is bounded as:*

$$C(\Pi_\epsilon) \leq C(\beta) \cdot \left(1 + \frac{\gamma}{(1-\gamma)f(\epsilon)}(1-\epsilon)\right)$$

where $f(\epsilon) \stackrel{\text{def}}{=} \min_{\boldsymbol{s} \in \mathcal{S}, \mu_\Pi(\boldsymbol{s}) > 0}[\mu(\boldsymbol{s})]$.

Proof. For notational clarity, we refer to Π_ϵ as Π in this proof. The term μ_Π is the highest discounted marginal state distribution starting from the initial state distribution ρ and following policies $\pi \in \Pi$. Formally, it is defined as:

$$\mu_\Pi \stackrel{\text{def}}{=} \max_{\{\pi_i\}_i:\ \forall\ i,\ \pi_i \in \Pi} (1-\gamma) \sum_{m=1}^{\infty} m\gamma^{m-1} \rho_0 P^{\pi_1} \cdots P^{\pi_m}$$

Now, we begin the proof of the theorem. We first note, from the definition of Π, $\forall\ \boldsymbol{s} \in \mathcal{S}\ \forall\ \pi \in \Pi, \pi(\boldsymbol{a}|\boldsymbol{s}) > 0 \implies \beta(\boldsymbol{a}|\boldsymbol{s}) > \epsilon$. This suggests a bound on the total variation distance between β and any $\pi \in \Pi$ for all $\boldsymbol{s} \in \mathcal{S}$, $D_{TV}(\beta(\cdot|\boldsymbol{s})||\pi(\cdot|\boldsymbol{s})) \leq 1 - \epsilon$. This means that the marginal state distributions of β and Π, are bounded in total variation distance by: $D_{TV}(\mu_\beta||\mu_\Pi) \leq \frac{\gamma}{1-\gamma}(1-\epsilon)$, where μ_Π is the marginal state distribution as defined above. This can be derived from Schulman et al. [289], Appendix B, which bounds the difference in returns of two policies by showing the state marginals between two policies are bounded if their total variation distance is bounded.

Further, the definition of the set of policies Π implies that $\forall\ s \in \mathcal{S}, \mu_\Pi(\boldsymbol{s}) > 0 \implies \mu_\beta(\boldsymbol{s}) \geq f(\epsilon)$, where $f(\epsilon) > 0$ is a constant that depends on ϵ and captures the minimum visitation probability of a state $\boldsymbol{s} \in \mathcal{S}$ when rollouts are executed from the initial state distribution ρ while executing the behaviour policy $\beta(\boldsymbol{a}|\boldsymbol{s})$, under the constraint that only actions with $\beta(\boldsymbol{a}|\boldsymbol{s}) \geq \epsilon$ are selected for execution in the environment. Combining it with the total variation divergence bound, $\max_s ||\mu_\beta(\boldsymbol{s}) - \mu_\Pi(\boldsymbol{s})|| \leq \frac{\gamma}{1-\gamma}(1-\epsilon)$, we get that

$$\sup_{\boldsymbol{s} \in \mathcal{S}} \frac{\mu_\Pi(\boldsymbol{s})}{\mu_\beta(\boldsymbol{s})} \leq 1 + \frac{\gamma}{(1-\gamma)f(\epsilon)}(1-\epsilon)$$

We know that, $C(\Pi) \stackrel{\text{def}}{=} (1-\gamma)^2 \sum_{k=1}^{\infty} k\gamma^{k-1} c(k)$ is the ratio of the marginal state visitation distribution under the policy iterates when performing backups using the distribution-constrained operator and the data distribution $\mu = \mu_\beta$. Therefore,

$$\frac{C(\Pi_\epsilon)}{C(\beta)} \stackrel{\text{def}}{=} \sup_{s \in \mathcal{S}} \frac{\mu_\Pi(s)}{\mu_\beta(s)} \leq 1 + \frac{\gamma}{(1-\gamma) f(\epsilon)} (1-\epsilon)$$

□

B.3 ADDITIONAL DETAILS REGARDING BEAR

In this appendix, we address several remaining points regarding the support matching formulation of BEAR, and further discuss its connections to prior work.

B.3.1 *Why can we choose actions from Π_ϵ, the support of the training distribution, and need not restrict action selection to the policy distribution?*

In Section 4.3, we designed a new distribution-constrained backup and analyzed its properties from an error propagation perspective. Theorems 4.3.4 and 4.3.5 tell us that, if the maximum projection error on all actions within the support of the train distribution is bounded, then the worst-case error incurred is also bounded. That is, we have a bound on $\max_{\pi \in \Pi_\epsilon} \mathbb{E}_\pi[\delta_k(s,a)]$. In this section, we provide an intuitive explanation for why action distributions that are very different from the training policy distributions, but still lie in the support of the train distribution, can be chosen without incurring large error. In practice, we use powerful function approximators for Q-learning, such as deep neural networks. That is, $\delta_k(s,a)$ is the Bellman error for one iteration of Q-iteration/Q-learning, which can essentially be viewed as a supervised regression problem with a very expressive function class. In this scenario, we expect a bounded error on the entire support of the training distribution, and we therefore expect approximation error to depend less on the specific density of a datapoint under the data distribution, and more on whether or not that datapoint is within the support of the data distribution. I.e., any point that is within the support would have a comparatively low error, due to the expressivity of the function approximator.

Another justification is that, a different version of the Bellman error objective renormalizes the action-distributions to the uniform distribution by applying an inverse behavior policy density weighting. For example, [15, 14] use this variant of Bellman error:

$$Q_{k+1} = \text{argmin}_Q \sum_{i=1, a_i \sim \beta(\cdot|s_i)}^{N} \frac{1}{\beta\left(a_i|s_i\right)} \left(Q\left(s_i, a_i\right) - \left[r(s,a) + \gamma \max_{a' \in \mathcal{A}} Q_k\left(s_{i+1}, a'\right) \right] \right)^2$$

This implies that this form of Bellman error mainly depends upon the support of the behaviour policy β (i.e. the set of action samples sampled from β with a high-enough

probability which we formally refer to as $\beta(\boldsymbol{a}|\boldsymbol{s}) \geq \epsilon$ in the main text). In a scenario when this form of Bellman error is being minimized, $\delta_k(\boldsymbol{s}, \boldsymbol{a})$ is defined as

$$\delta_k(\boldsymbol{s}, \boldsymbol{a}) = \frac{1}{\beta(\boldsymbol{a}|\boldsymbol{s})} |Q_k(\boldsymbol{s}, \boldsymbol{a}) - \mathcal{B}Q_{k-1}(\boldsymbol{s}, \boldsymbol{a})|$$

The overall error, hence, incurred due to error propagation is expected to be insensitive to distribution change, provided the support of the distribution doesn't change. Therefore, all policies $\pi \in \Pi_\epsilon$ incur the same amount of propagated error ($|V_k - V_\Pi|$) whereas different amount of suoptimality biases – suggesting the existence of a different policy in Π_ϵ which propagates the same amount of error while having a lower suboptimality bias. However, in practice, it has been observed that using the inverse density weighting under the behaviour policy doesn't lead to substantially better performance for vanilla RL (not in the setting with purely off-policy, static datasets), so we use the unmodified Bellman error objective.

Both of these justifications indicate that bounded $\delta_k(\boldsymbol{s}, \boldsymbol{a})$ is reasonable to expect under in-support action distributions.

B.3.2 *Details on connection between BEAR and distribution-constrained backups*

Distribution-constrained backups perform maximization over a set of policies Π_ϵ which is defined as the set of policies that share the support with the behaviour policy. In the BEAR-QL algorithm, π_ϕ is maximized towards maximizing the expected Q-value for each state under the action distribution defined by it, while staying in-support (through the MMD constraint). The maximization step biases π_ϕ towards the in-support actions which maximize the current Q-value. By sampling multiple Dirac-delta action distributions - $\delta_{\boldsymbol{a}_i}$ - and then performing an explicit maximum over them for computing the target is a stochastic approximation to the distribution-constrained operator. What is the importance of training the actor to maximize the expected Q-value? We found empirically that this step is important as without this maximization step and high-dimensional action spaces, it is likely to require many more samples (exponentially more, due to curse of dimensionality) to get the correct action that maximizes the target value while being in-support. This is hard and unlikely, and in some experiments we tried with this variant, we found it to lead to suboptimal solutions. At evaluation time, we use the Q-function as the actor. The same process is followed. Dirac-delta action distribution candidates $\delta_{\boldsymbol{a}_i}$ are sampled, and then the action $\boldsymbol{a}_i$ that is gives the empirical maximum over the Q-function values is the action that is executed in the environment.

B.3.3 *How effective is the* MMD *constraint in constraining supports of distributions?*

In Section 4.4, we argued in favour of the usage of the sampled MMD distance between distributions to search for a policy that is supported on the same support as the

train distribution. Revisiting the argument, in this section, we argue, via numerical simulations, the effectiveness of the MMD distance between two probability distributions in constraining the support of the distribution being learned, without constraining the distribution density function too much. While, MMD distance computed exactly between two distribution functions will match distributions exactly and that explains its applicability in 2-sample tests, however, with a limited number of samples, we empirically find that the values of the distance computed using samples from two d-dimensional Gaussian distributions with diagonal covariance matrices: $P \stackrel{\text{def}}{=} \mathcal{N}(\mu_P, \Sigma_P)$ and $Q \stackrel{\text{def}}{=} \mathcal{N}(\mu_Q, \Sigma_Q)$ is roughly equal to the distance computed using samples from $\mathcal{U}_\alpha(P) \stackrel{\text{def}}{=} [\text{ Uniform}(\mu_P^1 \pm \alpha\Sigma_P^{1,1})] \times \cdots \times [\text{ Uniform}(\mu_P^d \pm \alpha\Sigma_P^{d,d})]$ and Q. This means that when minimizing the distance to train distribution Q, the gradient signal would push Q towards a uniform distribution supported on P's support as this solution exhibits a lower MMD value – which is the objective we are optimizing.

Figure 16 shows an empirical comparison of $\text{MMD}(P, Q)$ when $Q = P$, computed by sampling n-samples from P, and $\text{MMD}(\mathcal{U}_\alpha(P), Q)$ (also when $Q = P$) computed by sampling n-samples from $\mathcal{U}_\alpha(P)$. We observe that distance computed using limited samples can, in fact, be higher between a distribution and itself as compared to a uniform distribution over a distribution's support and itself. In Figure 16, note that for smaller values of n and appropriately chosen α (mentioned against each figure, the support of the uniform distribution), the estimator for $(\mathcal{U}_\alpha(P), P)$ can provide lower estimates than the value of the estimator for (P, P). This observation suggests that when the number of samples is not enough to sample infer the distribution shape, density-agnostic distances like MMD can be used as optimization objectives to push distributions to match supports. Subfigures (c) and (d) shows the increase in MMD distance as the support of the uniform distribution is expanded.

B.4 Additional Experimental Details

Data collection. We trained behavior policies using the Soft Actor-Critic algorithm [125]. In all cases, random data was generated by running a uniform at random policy in the environment. Optimal data was generated by training SAC agents in all 4 domains until convergence to the returns mentioned in Figure 9. Mediocre data was generated by training a policy until the return value marked in each of the plots in Figure 7. Each of our datasets contained 1e6 samples. We used the same datasets for evaluating different algorithms to maintain uniformity across results.

Choice of kernels. In our experiments, we found that the choice of the kernel is an important design decision that needs to be made. In general, we found that a Laplacian kernel $k(x, y) = \exp(\frac{-||x-y||}{\sigma})$ worked well in all cases. Gaussian kernel $k(x, y) = \exp(\frac{-||x-y||^2}{2\sigma^2})$ worked quite well in the case of optimal dataset. For the Laplacian

kernel, we chose $\sigma = 10.0$ for Cheetah, Ant and Hopper, and $\sigma = 20.0$ for Walker. However, we found that $\sigma = 20.0$ worked well for all environments in all settings. For the Gaussian kernel, we chose $\sigma = 20.0$ for all settings. Kernels often tend to not provide relevant measurements of distance especially in high-dimensional spaces, so one direction for future work is to design right kernels. We further experimented with a mixture of Laplacian kernel with different bandwidth parameters σ $(1.0, 10.0, 50.0)$ on Hopper-v2 and Walker2d-v2 where we found that it performs comparably and sometimes is better than a simple Laplacian kernel, probably because it is able to track supports upto different levels of thresholds due to multiple kernels.

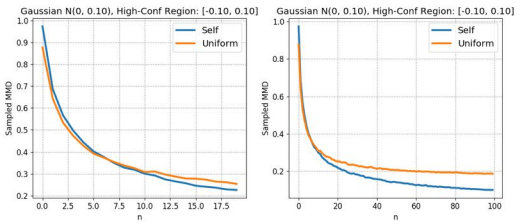
MORE DETAILS ABOUT THE ALGORITHM. At evaluation time, we find that using the greedy maximum of the Q-function over the support set of the behaviour policy (which can be approximated by sampling multiple Dirac-delta policies δ_{a_i} from the policy π_ϕ and performing a greedy maximization of the Q-values over these Dirac-delta policies) works best, better than unrolling the learned actor π_ϕ in the environment. This was also found useful in [95]. Another detail about the algorithm is deciding which samples to use for computing the objective. We train a parameteric model π_{data} which fits a tanh-Gaussian distribution to $\boldsymbol{a}$ given the states $\boldsymbol{s}$, $\pi_{data}(\cdot|\boldsymbol{s}) = \tanh\mathcal{N}(\mu(\cdot|\boldsymbol{s}), \sigma(\cdot|\boldsymbol{s}))$ and then use this to sample a candidate n actions for computing the MMD-distance, meaning that MMD is computed between $\boldsymbol{a}_1, \cdots, \boldsymbol{a}_N \sim \pi_{data}$ and π_ϕ. We find the latter to work better in practice. Also, computing the distance between actions before applying the tanh transformation work better, and leads to a constraint, that perhaps provides stronger gradient signal – because tanh saturates very quickly, after which gradients almost vanish.

OTHER HYPERPARAMETERS. Other hyperparameters include the following – (1) The variance of the Gaussian σ^2 /(standard deviation of) Laplacian kernel σ: We tried a variance of 10, 20, and 40. We found that 10 and 20 worked well across Cheetah, Hopper and Ant, and 20 worked well for Walker2d; (2) The learning rate for the Lagrange multiplier was chosen to be 1e-3, and the log of the Lagrange multiplier was clipped between $[-5, 10]$ to prevent instabilities; (3) For the policy improvement step, we found using average Q works better than min Q for Walker2d. For the baselines, we used BCQ code from the official implementation accompanying [95], TD3 code from the official implementation accompanying [96] and the BC baseline was the VAE-based behaviour cloning baseline also used in [95]. We evaluated on 10 evaluation episodes (which were separate from the train distribution) after every 1000 iterations and used the average score and the variance for the plots.

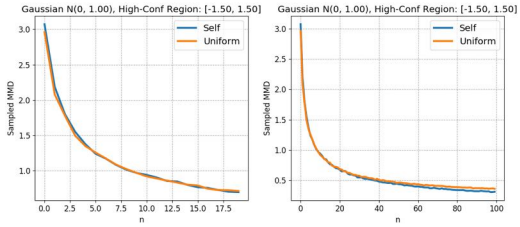
B.5 ADDITIONAL EXPERIMENTAL RESULTS

In this section, we provide some extra plots for some extra experiments. In Figure 17 we provide the difference between learned Q-values and Monte carlo returns of the policy in

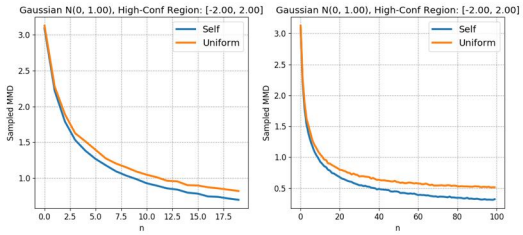
the environment. In Figure 18 we provide the trends of comparisons of Q-values learned by BEAR and BCQ in three environments.



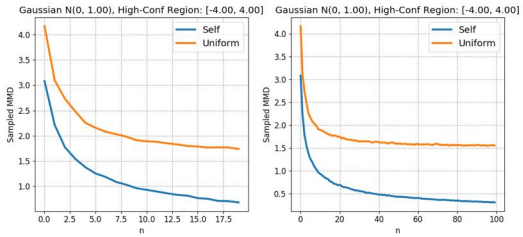
(a) $\mathcal{N}(0, 0.1), \mathcal{U}(-0.1, 0.1)$



(b) $\mathcal{N}(0, 1.0), \mathcal{U}(-1.5, 1.5)$



(c) $\mathcal{N}(0, 1.0), \mathcal{U}(-2.0, 2.0)$



(d) $\mathcal{N}(0, 1.0), \mathcal{U}(-4.0, 4.0)$

Figure 16: Comparing MMD distance between a 1-d Gaussian distribution (P) and itself (P), and a uniform distribution over support set of the P and the distribution P. The parameters of the Gaussian distribution (P) and the uniform distribution being considered are mentioned against each plot. ('Self' refers to (P, P) and 'Uniform' refers to $(P, \mathcal{U}(P))$.) Note that for small values of $n \approx 1 - 10$, the with the Uniform distribution is slightly lower in magnitude than the between the distribution P and itself (sub-figures (a), (b) and (c)). For (d), as the support of this uniform distribution is enlarged, this leads to an increase in the value of MMD in the uniform approximation case – which suggests that a near-local minimizer for the distance can be obtained by making sure that the distribution which is being trained in this process shares the same support as the other given distribution.

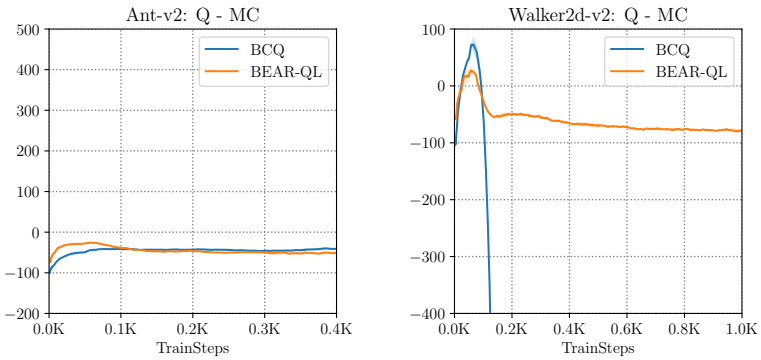


Figure 17: The trend of the difference between the Q-values and Monte-Carlo returns: $Q - MC$ returns for 2 environments. Note that a high value of $Q - MC$ corresponds to more overestimation. In these plots, BEAR-QL is more well behaved than BCQ. In Walker2d-v2, BCQ tends to diverge in the negative direction. In the case of Ant-v2, although roughly the same, the difference between Q values and Monte-carlo returns is slightly lower in the case of BEAR suggestion no risk of overestimation. (This corresponds to medium-quality data.)

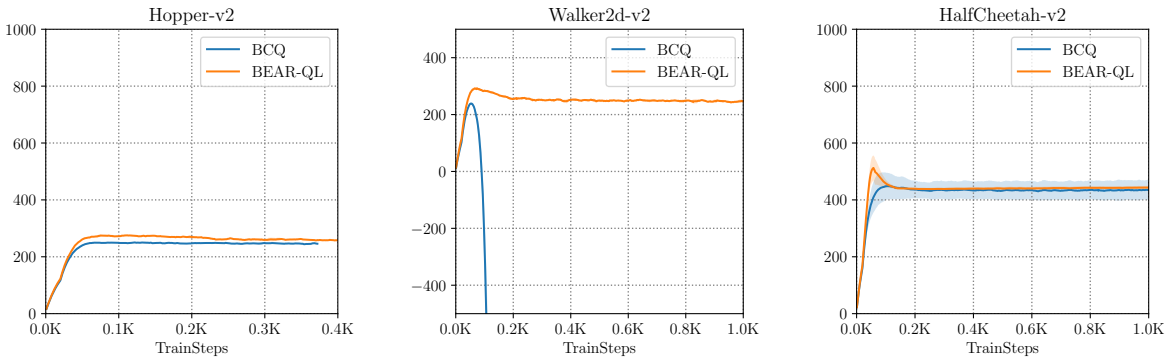


Figure 18: The trends of Q-values as a function of number of gradient steps taken in case of 3 environments. BCQs Q-values tend to be more unstable (especially in the case of Walker2d, where they diverge in the negative direction) as compared to BEAR. This corresponds to medium-quality data.

C

APPENDIX: CONSERVATIVE VALUE FUNCTION ESTIMATION

Model-Free Conservative Value Estimation

C.1 DISCUSSION OF CQL VARIANTS

We derive several variants of CQL in Section 5.1.3. Here, we discuss these variants on more detail and describe their specific properties. We first derive the variants: CQL($\mathcal{H}$), CQL(ρ), and then present another variant of CQL, which we call CQL(var). This third variant has strong connections to distributionally robust optimization [245].

CQL($\mathcal{H}$). In order to derive CQL($\mathcal{H}$), we substitute $\mathcal{R} = \mathcal{H}(\mu)$, and solve the optimization over μ in closed form for a given Q-function. For an optimization problem of the form:

$$\max_{\mu} \quad \mathbb{E}_{\boldsymbol{x}\sim\mu(\boldsymbol{x})}[f(\boldsymbol{x})] + \mathcal{H}(\mu) \quad \text{s.t.} \quad \sum_{\boldsymbol{x}} \mu(\boldsymbol{x}) = 1,\ \mu(\boldsymbol{x}) \geq 0\ \forall \boldsymbol{x},$$

the optimal solution is equal to $\mu^*(\boldsymbol{x}) = \frac{1}{Z}\exp(f(\boldsymbol{x}))$, where Z is a normalizing factor. Plugging this into Equation 5.1.3, we exactly obtain Equation 5.1.4.

CQL(ρ). In order to derive CQL(ρ), we follow the above derivation, but our regularizer is a KL-divergence regularizer instead of entropy.

$$\max_{\mu} \quad \mathbb{E}_{\boldsymbol{x}\sim\mu(\boldsymbol{x})}[f(\boldsymbol{x})] + D_{\text{KL}}(\mu||\rho) \quad \text{s.t.} \quad \sum_{\boldsymbol{x}} \mu(\boldsymbol{x}) = 1,\ \mu(\boldsymbol{x}) \geq 0\ \forall \boldsymbol{x}.$$

The optimal solution is given by, $\mu^*(\boldsymbol{x}) = \frac{1}{Z}\rho(\boldsymbol{x})\exp(f(\boldsymbol{x}))$, where Z is a normalizing factor. Plugging this back into the CQL family (Equation 5.1.3), we obtain the following objective for training the Q-function (modulo some normalization terms):

$$\min_Q\ \alpha\mathbb{E}_{\boldsymbol{s}\sim d^{\pi_\beta}(\boldsymbol{s})}\left[\mathbb{E}_{\boldsymbol{a}\sim\rho(\boldsymbol{a}|\boldsymbol{s})}\left[Q(\boldsymbol{s},\boldsymbol{a})\frac{\exp(Q(\boldsymbol{s},\boldsymbol{a}))}{Z'}\right] - \mathbb{E}_{\boldsymbol{a}\sim\pi_\beta(\boldsymbol{a}|\boldsymbol{s})}\left[Q(\boldsymbol{s},\boldsymbol{a})\right]\right] + \tfrac{1}{2}\mathbb{E}_{\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}'\sim\mathcal{D}}\left[\left(Q - \mathcal{B}^{\pi_k}\hat{Q}^k\right)^2\right]. \quad \text{(C.1.1)}$$

CQL(var). Finally, we derive a CQL variant that is inspired from the perspective of distributionally robust optimization (DRO) [245]. This version penalizes the variance in the Q-function across actions at all states $\boldsymbol{s}$, under some action-conditional distribution

of our choice. In order to derive a canonical form of this variant, we invoke an identity from Namkoong and Duchi [245], which helps us simplify Equation 5.1.3. To start, we define the notion of "robust expectation": for any function $f(\boldsymbol{x})$, and any empirical distribution $\hat{P}(\boldsymbol{x})$ over a dataset $\{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_N\}$ of N elements, the "robust" expectation defined by:

$$R_N(\hat{P}) := \max_{\mu(\boldsymbol{x})} \; \mathbb{E}_{\boldsymbol{x}\sim\mu(\boldsymbol{x})}[f(\boldsymbol{x})] \quad \text{s.t.} \quad D_f(\mu(\boldsymbol{x}), \hat{P}(\boldsymbol{x})) \leq \frac{\delta}{N},$$

can be approximated using the following upper-bound:

$$R_N(\hat{P}) \leq \mathbb{E}_{\boldsymbol{x}\sim\hat{P}(\boldsymbol{x})}[f(\boldsymbol{x})] + \sqrt{\frac{2\delta \; \mathrm{var}_{\hat{P}(\boldsymbol{x})}(f(\boldsymbol{x}))}{N}}, \tag{C.1.2}$$

where the gap between the two sides of the inequality decays inversely w.r.t. to the dataset size, $\mathcal{O}(1/N)$. By using Equation C.1.2 to simplify Equation 5.1.3, we obtain an objective for training the Q-function that penalizes the variance of Q-function predictions under the distribution $\hat{P}$.

$$\min_{Q} \; \frac{1}{2}\,\mathbb{E}_{\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}'\sim\mathcal{D}}\left[\left(Q - \mathcal{B}^{\pi_k}\hat{Q}^k\right)^2\right] + \alpha\mathbb{E}_{\boldsymbol{s}\sim d^{\pi_\beta}(\boldsymbol{s})}\left[\sqrt{\frac{\mathrm{var}_{\hat{P}(\boldsymbol{a}|\boldsymbol{s})}\left(Q(\boldsymbol{s},\boldsymbol{a})\right)}{d^{\pi_\beta}(\boldsymbol{s})|\mathcal{D}|}}\right] + \alpha\mathbb{E}_{s\sim d^{\pi_\beta}(\boldsymbol{s})}\left[\mathbb{E}_{\hat{P}(\boldsymbol{a}|\boldsymbol{s})}[Q(\boldsymbol{s},\boldsymbol{a})] - \mathbb{E}_{\pi_\beta(\boldsymbol{a}|\boldsymbol{s})}[Q(\boldsymbol{s},\boldsymbol{a})]\right] \tag{C.1.3}$$

The only remaining decision is the choice of $\hat{P}$, which can be chosen to be the inverse of the empirical action distribution in the dataset, $\hat{P}(\boldsymbol{a}|\boldsymbol{s}) \propto \frac{1}{\hat{D}(\boldsymbol{a}|\boldsymbol{s})}$, or even uniform over actions, $\hat{P}(\boldsymbol{a}|\boldsymbol{s}) = \mathrm{Unif}(\boldsymbol{a})$, to obtain this variant of variance-regularized CQL.

C.2 Discussion of gap-expanding behavior of CQL backups

In this section, we discuss in detail the consequences of the gap-expanding behavior of CQL backups over prior methods based on policy constraints that, as we show in this section, may not exhibit such gap-expanding behavior in practice. To recap, Theorem 5.1.4 shows that the CQL backup operator increases the difference between expected Q-value at in-distribution ($\boldsymbol{a} \sim \pi_\beta(\boldsymbol{a}|\boldsymbol{s})$) and out-of-distribution ($\boldsymbol{a}$ s.t. $\frac{\mu_k(\boldsymbol{a}|\boldsymbol{s})}{\pi_\beta(\boldsymbol{a}|\boldsymbol{s})} << 1$) actions. We refer to this property as the gap-expanding property of the CQL update operator.

Function approximation may give rise to erroneous Q-values at OOD actions. We start by discussing the behavior of prior methods based on policy constraints [179, 95, 149, 343] in the presence of function approximation. To recap, because computing the target value requires $\mathbb{E}_\pi[\hat{Q}(\boldsymbol{s},\boldsymbol{a})]$, constraining π to be close to π_β will avoid evaluating $\hat{Q}$ on OOD actions. These methods typically do not impose any further form of regularization on

the learned Q-function. Even with policy constraints, because function approximation used to represent the Q-function, learned Q-values at two distinct state-action pairs are coupled together. As prior work has argued and shown [4, 90, 180], the "generalization" or the coupling effects of the function approximator may be heavily influenced by the properties of the data distribution [90, 180]. For instance, Fu et al. [90] empirically shows that when the dataset distribution is narrow (i.e. state-action marginal entropy, $\mathcal{H}(d^{\pi_\beta}(\boldsymbol{s}, \boldsymbol{a}))$, is low [90]), the coupling effects of the Q-function approximator can give rise to incorrect Q-values at different states, though this behavior is absent without function approximation, and is not as severe with high-entropy (e.g. Uniform) state-action marginal distributions.

In offline RL, we will shortly present empirical evidence on high-dimensional MuJoCo tasks showing that certain dataset distributions, $\mathcal{D}$, may cause the learned Q-value at an OOD action $\boldsymbol{a}$ at a state $\boldsymbol{s}$, to in fact take on high values than Q-values at in-distribution actions at intermediate iterations of learning. This problem persists even when a large number of samples (e.g. $1M$) are provided for training, and the agent cannot correct these errors due to no active data collection.

Since actor-critic methods, including those with policy constraints, use the learned Q-function to train the policy, in an iterative online policy evaluation and policy improvement cycle, the errneous Q-function may push the policy towards OOD actions, especially when no policy constraints are used. Of course, policy constraints should prevent the policy from choosing OOD actions, however, as we will show that in certain cases, policy constraint methods might also fail to prevent the effects on the policy due to incorrectly high Q-values at OOD actions.

How can CQL address this problem? As we show in Theorem 5.1.4, the difference between expected Q-values at in-distribution actions and out-of-distribution actions is expanded by the CQL update. This property is a direct consequence of the specific nature of the CQL regularizer – that maximizes Q-values under the dataset distribution, and minimizes them otherwise. This difference depends upon the choice of α_k, which can directly be controlled, since it is a free parameter. Thus, by effectively controlling α_k, CQL can push down the learned Q-value at out-of-distribution actions as much is desired, correcting for the erroneous overestimation error in the process.

Empirical evidence on high-dimensional benchmarks with neural networks. We next empirically demonstrate the existence of of such Q-function estimation error on high-dimensional MuJoCo domains when deep neural network function approximators are used with stochastic optimization techniques. In order to measure this error, we plot the difference in expected Q-value under actions sampled from the behavior distribution,

$\boldsymbol{a} \sim \pi_\beta(\boldsymbol{a}|\boldsymbol{s})$, and the maximum Q-value over actions sampled from a uniformly random policy, $\boldsymbol{a} \sim \text{Unif}(\boldsymbol{a}|\boldsymbol{s})$. That is, we plot the quantity

$$\hat{\Delta}^k = \mathbb{E}_{\boldsymbol{s},\boldsymbol{a}\sim\mathcal{D}} \left[\max_{\boldsymbol{a}'_1,\cdots,\boldsymbol{a}'_N \sim \text{Unif}(\boldsymbol{a}')} [\hat{Q}^k(\boldsymbol{s},\boldsymbol{a}')] - \hat{Q}^k(\boldsymbol{s},\boldsymbol{a}) \right] \tag{C.2.1}$$

over the iterations of training, indexed by k. This quantity, intuitively, represents an estimate of the "advantage" of an action $\boldsymbol{a}$, under the Q-function, with respect to the optimal action $\max_{\boldsymbol{a}'} \hat{Q}^k(\boldsymbol{s},\boldsymbol{a}')$. Since, we cannot perform exact maximization over the learned Q-function in a continuous action space to compute Δ, we estimate it via sampling described in Equation C.2.1.

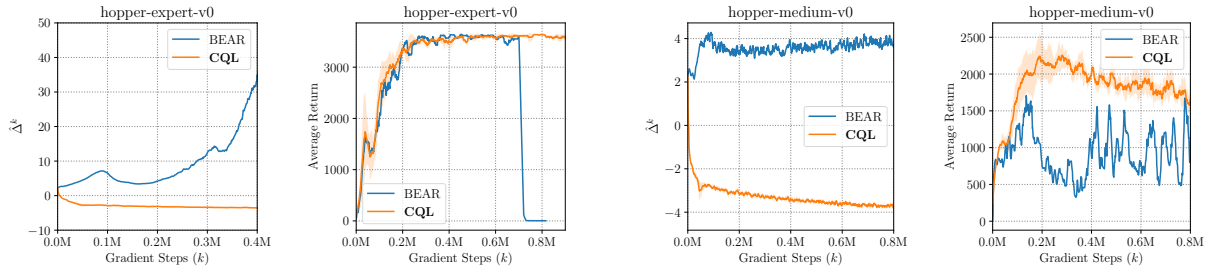
We present these plots in Figure 19 on two datasets: hopper-expert and hopper-medium. The expert dataset is generated from a near-deterministic, expert policy, exhibits a narrow coverage of the state-action space, and limited to only a few directed trajectories. On this dataset, we find that $\hat{\Delta}^k$ is always positive for the policy constraint method (Figure 19(a)) and increases during training – note, the continuous rise in $\hat{\Delta}^k$ values, in the case of the policy-constraint method, shown in Figure 19(a). This means that even if the dataset is generated from an expert policy, and policy constraints correct target values for OOD actions, incorrect Q-function generalization may make an out-of-distribution action appear promising. For the more stochastic hopper-medium dataset, that consists of a more diverse set of trajectories, shown in Figure 19(b), we still observe that $\hat{\Delta}^k > 0$ for the policy-constraint method, however, the relative magnitude is smaller than hopper-expert.

In contrast, Q-functions learned by CQL, generally satisfy $\hat{\Delta}^k < 0$, as is seen and these values are clearly smaller than those for the policy-constraint method. This provides some empirical evidence for Theorem 5.1.4, in that, the maximum Q-value at a randomly chosen action from the uniform distribution the action space is smaller than the Q-value at in-distribution actions.

On the hopper-expert task, as we show in Figure 19(a) (right), we eventually observe an "unlearning" effect, in the policy-constraint method where the policy performance deteriorates after a extra iterations in training. This "unlearning" effect is similar to what has been observed when standard off-policy Q-learning algorithms without any policy constraint are used in the offline regime [179, 204], on the other hand this effect is absent in the case of CQL, even after equally many training steps. The performance in the more-stochastic hopper-medium dataset fluctuates, but does not deteriorate.

To summarize this discussion, we concretely observed the following points via empirical evidence:

- CQL backups are gap expanding in practice, as justified by the negative $\hat{\Delta}^k$ values in Figure 19.



(a) hopper-expert-vo **(b)** hopper-medium-vo

Figure 19: Δ^k as a function of training iterations for hopper-expert and hopper-medium datasets. Note that CQL (left) generally has negative values of Δ, whereas BEAR (right) generally has positive Δ values, which also increase during training with increasing k values.

- Policy constraint methods, that do not impose any regularization on the Q-function may observe highly positive $\hat{\Delta}^k$ values during training, especially with narrow data distributions, indicating that gap-expansion may be absent.
- When $\hat{\Delta}^k$ values continuously grow during training, the policy might eventually suffer from an unlearning effect [204], as shown in Figure 19(a).

C.3 THEOREM PROOFS

In this section, we provide proofs of the theorems in Sections 5.1.2 and 5.1.3. We first redefine notation for clarity and then provide the proofs of the results in the main paper.

Notation. Let $k \in \mathbb{N}$ denote an iteration of policy evaluation (in Section 5.1.2) or Q-iteration (in Section 5.1.3). In an iteration k, the objective – Equation 5.1.2 or Equation 5.1.3 – is optimized using the previous iterate (i.e. $\hat{Q}^{k-1}$) as the target value in the backup. Q^k denotes the true, tabular Q-function iterate in the MDP, without any correction. In an iteration, say $k+1$, the current tabular Q-function iterate, Q^{k+1} is related to the previous tabular Q-function iterate Q^k as: $Q^{k+1} = \mathcal{B}^\pi Q^k$ (for policy evaluation) or $Q^{k+1} = \mathcal{B}^{\pi_k} Q^k$ (for policy learning). Let $\hat{Q}^k$ denote the k-th Q-function iterate obtained from CQL. Let $\hat{V}^k$ denote the value function, $\hat{V}^k := \mathbb{E}_{\boldsymbol{a} \sim \pi(\boldsymbol{a}|\boldsymbol{s})}[\hat{Q}^k(\boldsymbol{s}, \boldsymbol{a})]$.

A note on the value of α. Before proving the theorems, we remark that while the statements of Theorems 5.1.2, 5.1.1 and C.4.1 (we discuss this in Appendix C.4) show that CQL produces lower bounds if α is larger than some threshold, so as to overcome either sampling error (Theorems 5.1.2 and 5.1.1) or function approximation error (Theorem C.4.1). While the optimal α_k in some of these cases depends on the current Q-value, $\hat{Q}^k$, we can always choose a worst-case value of α_k by using the inequality $\hat{Q}^k \leq 2R_{\max}/(1-\gamma)$, still guaranteeing a lower bound. If it is unclear why the learned Q-function $\hat{Q}^k$ should be bounded, we can always clamp the Q-values if they go outside $\left[\frac{-2R_{\max}}{1-\gamma}, \frac{2R_{\max}}{1-\gamma}\right]$. We first prove Theorem 5.1.1, which shows that policy evaluation using a simplified version of CQL (Equation 5.1.1) results in a point-wise lower-bound on the Q-function.

Proof of Theorem 5.1.1

We start by analyzing $\hat{Q}^k$ by setting the derivative of Equation 5.1.1 to 0:

$$\forall\ \boldsymbol{s}, \boldsymbol{a} \in \mathcal{D}, k, \quad \hat{Q}^{k+1}(\boldsymbol{s}, \boldsymbol{a}) = \hat{\mathcal{B}}^{\pi}\hat{Q}^{k}(\boldsymbol{s}, \boldsymbol{a}) - \alpha \frac{\mu(\boldsymbol{a}|\boldsymbol{s})}{\hat{\pi}_{\beta}(\boldsymbol{a}|\boldsymbol{s})}. \tag{C.3.1}$$

Now, since, $\mu(\boldsymbol{a}|\boldsymbol{s}) > 0, \alpha > 0, \hat{\pi}_{\beta}(\boldsymbol{a}|\boldsymbol{s}) > 0$, we observe that at each iteration we underestimate the next Q-value iterate, i.e. $\hat{Q}^{k+1} \leq \hat{\mathcal{B}}^{\pi}\hat{Q}^{k}$.

Accounting for sampling error. Note that so far we have only shown that the Q-values are upper-bounded by the the "empirical Bellman targets" given by, $\hat{\mathcal{B}}^{\pi}\hat{Q}^{k}$. In order to relate $\hat{Q}^k$ to the true Q-value iterate, Q^k, we need to relate the empirical Bellman operator, $\hat{\mathcal{B}}^{\pi}$ to the actual Bellman operator, $\mathcal{B}^{\pi}$. In Appendix C.4.3, we show that if the reward function $r(\boldsymbol{s}, \boldsymbol{a})$ and the transition function, $T(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a})$ satisfy "concentration" properties, meaning that the difference between the observed reward sample, $r\ (\boldsymbol{s}, \boldsymbol{a}, r, \boldsymbol{s}') \in \mathcal{D})$ and the actual reward function $r(\boldsymbol{s}, \boldsymbol{a})$ (and analogously for the transition matrix) is bounded with high probability, then overestimation due to the empirical Backup operator is bounded. Formally, with high probability (w.h.p.) $\geq 1 - \delta$, $\delta \in (0, 1)$,

$$\forall Q, \boldsymbol{s}, \boldsymbol{a} \in \mathcal{D}, \quad \left|\hat{\mathcal{B}}^{\pi}Q(\boldsymbol{s}, \boldsymbol{a}) - \mathcal{B}^{\pi}Q(\boldsymbol{s}, \boldsymbol{a})\right| \leq \frac{C_{r,T,\delta}R_{\max}}{(1-\gamma)\sqrt{|\mathcal{D}(\boldsymbol{s}, \boldsymbol{a})|}}.$$

Hence, the following can be obtained, w.h.p.:

$$\hat{Q}^{k+1}(\boldsymbol{s}, \boldsymbol{a}) = \mathcal{B}^{\pi}\hat{Q}^{k}(\boldsymbol{s}, \boldsymbol{a}) \leq \mathcal{B}^{\pi}\hat{Q}^{k}(\boldsymbol{s}, \boldsymbol{a}) - \alpha \frac{\mu(\boldsymbol{a}|\boldsymbol{s})}{\hat{\pi}_{\beta}(\boldsymbol{a}|\boldsymbol{s})} + \frac{C_{r,T,\delta}R_{\max}}{(1-\gamma)\sqrt{|\mathcal{D}(\boldsymbol{s}, \boldsymbol{a})|}}. \tag{C.3.2}$$

Now we need to reason about the fixed point of the update procedure in Equation C.3.1. The fixed point of Equation C.3.1 is given by:

$$\hat{Q}^{\pi} \leq \mathcal{B}^{\pi}\hat{Q}^{\pi} - \alpha \frac{\mu(\boldsymbol{a}|\boldsymbol{s})}{\hat{\pi}_{\beta}(\boldsymbol{a}|\boldsymbol{s})} + \frac{C_{r,T,\delta}R_{\max}}{(1-\gamma)\sqrt{|\mathcal{D}(\boldsymbol{s}, \boldsymbol{a})|}}$$

$$\implies \hat{Q}^{\pi} \leq (I - \gamma P^{\pi})^{-1}\left[R - \alpha \frac{\mu}{\hat{\pi}_{\beta}} + \frac{C_{r,T,\delta}R_{\max}}{1-\gamma)\sqrt{|\mathcal{D}}}\right]$$

$$\hat{Q}^{\pi}(\boldsymbol{s}, \boldsymbol{a}) \leq Q^{\pi}(\boldsymbol{s}, \boldsymbol{a}) - \alpha\left[(I - \gamma P^{\pi})^{-1}\left[\frac{\mu}{\hat{\pi}_{\beta}}\right]\right](\boldsymbol{s}, \boldsymbol{a}) + \left[(I - \gamma P^{\pi})^{-1}\frac{C_{r,T,\delta}R_{\max}}{(1-\gamma)\sqrt{|\mathcal{D}|}}\right](\boldsymbol{s}, \boldsymbol{a}),$$

thus proving the relationship in Theorem 5.1.1.

In order to guarantee a lower bound, α can be chosen to cancel any potential overestimation incurred by $\frac{C_{r,T,\delta}R_{\max}}{(1-\gamma)\sqrt{|\mathcal{D}|}}$. Note that this choice works, since $(I - \gamma P^{\pi})^{-1}$ is a matrix

with all non-negative entries. The choice of α that guarantees a lower bound is then given by:

$$\alpha \cdot \min_{s,a} \left[\frac{\mu(a|s)}{\hat{\pi}_\beta(a|s)} \right] \geq \max_{s,a} \frac{C_{r,T,\delta} R_{\max}}{(1-\gamma)\sqrt{|\mathcal{D}(s,a)|}}$$

$$\implies \alpha \geq \max_{s,a} \frac{C_{r,T,\delta} R_{\max}}{(1-\gamma)\sqrt{|\mathcal{D}(s,a)|}} \cdot \max_{s,a} \left[\frac{\mu(a|s)}{\hat{\pi}_\beta(a|s)} \right]^{-1}.$$

Note that the theoretically minimum possible value of α decreases as more samples are observed, i.e., when $|\mathcal{D}(s,a)|$ is large. Also, note that since, $\frac{C_{r,T,\delta} R_{\max}}{(1-\gamma 0\sqrt{|\mathcal{D}|}} \approx 0$, when $\hat{\mathcal{B}}^\pi = \mathcal{B}^\pi$, any $\alpha \geq 0$ guarantees a lower bound. And so choosing a value of $\alpha = 0$ is sufficient in this case.

Next, we prove Theorem 5.1.3 that shows that the additional term that maximizes the expected Q-value under the dataset distribution, $\mathbb{D}(s,a)$, (or $d^{\pi_\beta}(s)\pi_\beta(a|s)$, in the absence of sampling error), results in a lower-bound on only the expected value of the policy at a state, and not a pointwise lower-bound on Q-values at all actions.

Proof of Theorem 5.1.2

We first prove this theorem in the absence of sampling error, and then incorporate sampling error at the end, using a technique similar to the previous proof. In the tabular setting, we can set the derivative of the modified objective in Equation 5.1.2, and compute the Q-function update induced in the exact, tabular setting (this assumes $\hat{\mathcal{B}}^\pi = \mathcal{B}^\pi$) and $\pi_\beta(a|s) = \hat{\pi}_\beta(a|s)$).

$$\forall\ s, a, k \quad \hat{Q}^{k+1}(s,a) = \mathcal{B}^\pi \hat{Q}^k(s,a) - \alpha \left[\frac{\mu(a|s)}{\pi_\beta(a|s)} - 1 \right]. \tag{C.3.3}$$

Note that for state-action pairs, (s,a), such that, $\mu(a|s) < \pi_\beta(a|s)$, we are infact adding a positive quantity, $1 - \frac{\mu(a|s)}{\pi_\beta(a|s)}$, to the Q-function obtained, and this we cannot guarantee a point-wise lower bound, i.e. $\exists\ s, a,$ s.t. $\hat{Q}^{k+1}(s,a) \geq Q^{k+1}(s,a)$. To formally prove this, we can construct a counter-example three-state, two-action MDP, and choose a specific behavior policy $\pi(a|s)$, such that this is indeed the case.

The value of the policy, on the other hand, $\hat{V}^{k+1}$ is underestimated, since:

$$\hat{V}^{k+1}(s) := \mathbb{E}_{a \sim \pi(a|s)} \left[\hat{Q}^{k+1}(s,a) \right] = \mathcal{B}^\pi \hat{V}^k(s) - \alpha \mathbb{E}_{a \sim \pi(a|s)} \left[\frac{\mu(a|s)}{\pi_\beta(a|s)} - 1 \right]. \tag{C.3.4}$$

and we can show that $D_{\text{CQL}}(\boldsymbol{s}) := \sum_{\boldsymbol{a}} \pi(\boldsymbol{a}|\boldsymbol{s}) \left[\frac{\mu(\boldsymbol{a}|\boldsymbol{s})}{\pi_\beta(\boldsymbol{a}|\boldsymbol{s})} - 1\right]$ is always positive, when $\pi(\boldsymbol{a}|\boldsymbol{s}) = \mu(\boldsymbol{a}|\boldsymbol{s})$. To note this, we present the following derivation:

$$
\begin{aligned}
D_{\text{CQL}}(\boldsymbol{s}) :=& \sum_{\boldsymbol{a}} \pi(\boldsymbol{a}|\boldsymbol{s}) \left[\frac{\mu(\boldsymbol{a}|\boldsymbol{s})}{\pi_\beta(\boldsymbol{a}|\boldsymbol{s})} - 1\right] \\
&= \sum_{\boldsymbol{a}} (\pi(\boldsymbol{a}|\boldsymbol{s}) - \pi_\beta(\boldsymbol{a}|\boldsymbol{s}) + \pi_\beta(\boldsymbol{a}|\boldsymbol{s})) \left[\frac{\mu(\boldsymbol{a}|\boldsymbol{s})}{\pi_\beta(\boldsymbol{a}|\boldsymbol{s})} - 1\right] \\
&= \sum_{\boldsymbol{a}} (\pi(\boldsymbol{a}|\boldsymbol{s}) - \pi_\beta(\boldsymbol{a}|\boldsymbol{s})) \left[\frac{\pi(\boldsymbol{a}|\boldsymbol{s}) - \pi_\beta(\boldsymbol{a}|\boldsymbol{s})}{\pi_\beta(\boldsymbol{a}|\boldsymbol{s}}\right] + \sum_{\boldsymbol{a}} \pi_\beta(\boldsymbol{a}|\boldsymbol{s}) \left[\frac{\mu(\boldsymbol{a}|\boldsymbol{s})}{\pi_\beta(\boldsymbol{a}|\boldsymbol{s})} - 1\right] \\
&= \underbrace{\sum_{\boldsymbol{a}} \left[\frac{\left(\pi(\boldsymbol{a}|\boldsymbol{s}) - \pi_\beta(\boldsymbol{a}|\boldsymbol{s})\right)^2}{\pi_\beta(\boldsymbol{a}|\boldsymbol{s})}\right]}_{\geq 0} + 0 \quad \text{since, } \sum_{\boldsymbol{a}} \pi(\boldsymbol{a}|\boldsymbol{s}) = \sum_{\boldsymbol{a}} \pi_\beta(\boldsymbol{a}|\boldsymbol{s}) = 1.
\end{aligned}
$$

Note that the marked term, is positive since both the numerator and denominator are positive, and this implies that $D_{\text{CQL}}(\boldsymbol{s}) \geq 0$. Also, note that $D_{\text{CQL}}(\boldsymbol{s}) = 0$, iff $\pi(\boldsymbol{a}|\boldsymbol{s}) = \pi_\beta(\boldsymbol{a}|\boldsymbol{s})$. This implies that each value iterate incurs some underestimation, $\hat{V}^{k+1}(\boldsymbol{s}) \leq \mathcal{B}^\pi \hat{V}^k(\boldsymbol{s})$.

Now, we can compute the fixed point of the recursion in Equation C.3.4, and this gives us the following estimated policy value:

$$
\hat{V}^\pi(\boldsymbol{s}) = V^\pi(\boldsymbol{s}) - \alpha \left[\underbrace{(I - \gamma P^\pi)^{-1}}_{\text{non-negative entries}} \underbrace{\mathbb{E}_\pi \left[\frac{\pi}{\pi_\beta} - 1\right]}_{\geq 0}\right](\boldsymbol{s}),
$$

thus showing that in the absence of sampling error, Theorem 5.1.2 gives a lower bound. It is straightforward to note that this expression is tighter than the expression for policy value in Proposition 5.1.2, since, we explicitly subtract 1 in the expression of Q-values from the previous proof.

Incorporating sampling error. To extend this result to the setting with sampling error, similar to the previous result, the maximal overestimation at each iteration k, is bounded by $\frac{C_{r,T,\delta} R_{\max}}{1-\gamma}$. The resulting value-function satisfies (w.h.p.), $\forall \boldsymbol{s} \in \mathcal{D}$,

$$
\hat{V}^\pi(\boldsymbol{s}) \leq V^\pi(\boldsymbol{s}) - \alpha \left[(I - \gamma P^\pi)^{-1} \mathbb{E}_\pi \left[\frac{\pi}{\hat{\pi}_\beta} - 1\right]\right](\boldsymbol{s}) + \left[(I - \gamma P^\pi)^{-1} \frac{C_{r,T,\delta} R_{\max}}{(1-\gamma)\sqrt{|\mathcal{D}|}}\right](\boldsymbol{s})
$$

thus proving the theorem statement. In this case, the choice of α, that prevents overestimation w.h.p. is given by:

$$\alpha \geq \max_{s,a \in \mathcal{D}} \frac{C_{r,T} R_{\max}}{(1-\gamma)\sqrt{|\mathcal{D}(s,a)|}} \cdot \max_{s \in \mathcal{D}} \left[\sum_{a} \pi(a|s) \left(\frac{\pi(a|s)}{\hat{\pi}_\beta(a|s))} - 1 \right) \right]^{-1}.$$

Similar to Theorem 5.1.1, note that the theoretically acceptable value of α decays as the number of occurrences of a state action pair in the dataset increases. Next we provide a proof for Theorem 5.1.3.

Proof of Theorem 5.1.3

In order to prove this theorem, we compute the difference induced in the policy value, $\hat{V}^{k+1}$, derived from the Q-value iterate, $\hat{Q}^{k+1}$, with respect to the previous iterate $\mathcal{B}^\pi \hat{Q}^k$. If this difference is negative at each iteration, then the resulting Q-values are guaranteed to lower bound the true policy value.

$$\begin{aligned}
\mathbb{E}_{\hat{\pi}^{k+1}(a|s)}[\hat{Q}^{k+1}(s,a)] &= \mathbb{E}_{\hat{\pi}^{k+1}(a|s)} \left[\mathcal{B}^\pi \hat{Q}^k(s,a) \right] - \mathbb{E}_{\hat{\pi}^{k+1}(a|s)} \left[\frac{\pi_{\hat{Q}^k}(a|s)}{\hat{\pi}_\beta(a|s)} - 1 \right] \\
&= \mathbb{E}_{\hat{\pi}^{k+1}(a|s)} \left[\mathcal{B}^\pi \hat{Q}^k(s,a) \right] - \underbrace{\mathbb{E}_{\pi_{\hat{Q}^k}(a|s)} \left[\frac{\pi_{\hat{Q}^k}(a|s)}{\hat{\pi}_\beta(a|s)} - 1 \right]}_{\text{underestimation, (a)}} \\
&\quad + \sum_{a} \underbrace{\left(\pi_{\hat{Q}^k}(a|s) - \hat{\pi}^{k+1}(a|s) \right)}_{\text{(b), } \leq \mathrm{D_{TV}}(\pi_{\hat{Q}^k}, \hat{\pi}^{k+1})} \frac{\pi_{\hat{Q}^k}(a|s)}{\hat{\pi}_\beta(a|s)}
\end{aligned}$$

If (a) has a larger magnitude than (b), then the learned Q-value induces an underestimation in an iteration $k+1$, and hence, by a recursive argument, the learned Q-value underestimates the optimal Q-value. We note that by upper bounding term (b) by $\mathrm{D_{TV}}(\pi_{\hat{Q}^k}, \hat{\pi}^{k+1}) \cdot \max_a \frac{\pi_{\hat{Q}^k}(a|s)}{\hat{\pi}_\beta(a|s)}$, and writing out (a) ¿ upper-bound on (b), we obtain the desired result.

Finally, we show that under specific choices of $\alpha_1, \cdots, \alpha_k$, the CQL backup is gap-expanding by providing a proof for Theorem 5.1.4.

Proof of Theorem 5.1.4 (CQL is gap-expanding)

For this theorem, we again first present the proof in the absence of sampling error, and then incorporate sampling error into the choice of α. We follow the strategy of observing the Q-value update in one iteration. Recall that the expression for the Q-value iterate at iteration k is given by:

$$\hat{Q}^{k+1}(\boldsymbol{s}, \boldsymbol{a}) = \mathcal{B}^{\pi^k} \hat{Q}^k(\boldsymbol{s}, \boldsymbol{a}) - \alpha_k \frac{\mu_k(\boldsymbol{a}|\boldsymbol{s}) - \pi_\beta(\boldsymbol{a}|\boldsymbol{s})}{\pi_\beta(\boldsymbol{a}|\boldsymbol{s})}.$$

Now, the value of the policy $\mu_k(\boldsymbol{a}|\boldsymbol{s})$ under $\hat{Q}^{k+1}$ is given by:

$$\mathbb{E}_{\boldsymbol{a}\sim\mu_k(\boldsymbol{a}|\boldsymbol{s})}[\hat{Q}^{k+1}(\boldsymbol{s}, \boldsymbol{a})] = \mathbb{E}_{\boldsymbol{a}\sim\mu_k(\boldsymbol{a}|\boldsymbol{s})}[\mathcal{B}^{\pi^k} \hat{Q}^k(\boldsymbol{s}, \boldsymbol{a})] - \alpha_k \underbrace{\mu_k^T \left(\frac{\mu_k(\boldsymbol{a}|\boldsymbol{s}) - \pi_\beta(\boldsymbol{a}|\boldsymbol{s})}{\pi_\beta(\boldsymbol{a}|\boldsymbol{s})} \right)}_{:=\hat{\Delta}^k, \geq 0, \text{ by proof of Theorem 5.1.2.}}$$

Now, we also note that the expected amount of extra underestimation introduced at iteration k under action sampled from the behavior policy $\pi_\beta(\boldsymbol{a}|\boldsymbol{s})$ is o, as,

$$\mathbb{E}_{\boldsymbol{a}\sim\pi_\beta(\boldsymbol{a}|\boldsymbol{s})}[\hat{Q}^{k+1}(\boldsymbol{s}, \boldsymbol{a})] = \mathbb{E}_{\boldsymbol{a}\sim\pi_\beta(\boldsymbol{a}|\boldsymbol{s})}[\mathcal{B}^{\pi^k} \hat{Q}^k(\boldsymbol{s}, \boldsymbol{a})] - \alpha_k \underbrace{\pi_\beta{}^T \left(\frac{\mu_k(\boldsymbol{a}|\boldsymbol{s}) - \pi_\beta(\boldsymbol{a}|\boldsymbol{s})}{\pi_\beta(\boldsymbol{a}|\boldsymbol{s})} \right)}_{=0}.$$

where the marked quantity is equal to o since it is equal since $\pi_\beta(\boldsymbol{a}|\boldsymbol{s})$ in the numerator cancels with the denominator, and the remaining quantity is a sum of difference between two density functions, $\sum_{\boldsymbol{a}} \mu_k(\boldsymbol{a}|\boldsymbol{s}) - \pi_\beta(\boldsymbol{a}|\boldsymbol{s})$, which is equal to o. Thus, we have shown that,

$$\mathrm{E}_{\pi_\beta(\boldsymbol{a}|\boldsymbol{s})}[\hat{Q}^{k+1}(\boldsymbol{s}, \boldsymbol{a})] - \mathbb{E}_{\mu_k(\boldsymbol{a}|\boldsymbol{s})}[\hat{Q}^{k+1}(\boldsymbol{s}, \boldsymbol{a})] = \mathbb{E}_{\pi_\beta(\boldsymbol{a}|\boldsymbol{s})}[\mathcal{B}^{\pi^k} \hat{Q}^k(\boldsymbol{s}, \boldsymbol{a})] - \mathbb{E}_{\mu_k(\boldsymbol{a}|\boldsymbol{s})}[\mathcal{B}^{\pi^k} \hat{Q}^k(\boldsymbol{s}, \boldsymbol{a})] - \alpha_k \hat{\Delta}^k.$$

Now subtracting the difference, $\mathbb{E}_{\pi_\beta(\boldsymbol{a}|\boldsymbol{s})}[Q^{k+1}(\boldsymbol{s}, \boldsymbol{a})] - \mathbb{E}_{\mu_k(\boldsymbol{a}|\boldsymbol{s})}[Q^{k+1}(\boldsymbol{s}, \boldsymbol{a})]$, computed under the tabular Q-function iterate, Q^{k+1}, from the previous equation, we obtain that

$$\begin{aligned}\mathbb{E}_{\boldsymbol{a}\sim\pi_\beta(\boldsymbol{a}|\boldsymbol{s})}[\hat{Q}^{k+1}(\boldsymbol{s}, \boldsymbol{a})] - \mathbb{E}_{\pi_\beta(\boldsymbol{a}|\boldsymbol{s})}[Q^{k+1}(\boldsymbol{s}, \boldsymbol{a})] = \mathbb{E}_{\mu_k(\boldsymbol{a}|\boldsymbol{s})}[\hat{Q}^{k+1}(\boldsymbol{s}, \boldsymbol{a})] - \mathbb{E}_{\mu_k(\boldsymbol{a}|\boldsymbol{s})}[Q^{k+1}(\boldsymbol{s}, \boldsymbol{a})] \\ + (\mu_k(\boldsymbol{a}|\boldsymbol{s}) - \pi_\beta(\boldsymbol{a}|\boldsymbol{s}))^T \underbrace{\left[\mathcal{B}^{\pi^k} \left(\hat{Q}^k - Q^k \right)(\boldsymbol{s}, \cdot) \right]}_{(a)} - \alpha_k \hat{\Delta}^k.\end{aligned}$$

Now, by choosing α_k, such that any positive bias introduced by the quantity $(\mu_k(\boldsymbol{a}|\boldsymbol{s}) - \pi_\beta(\boldsymbol{a}|\boldsymbol{s}))^T(a)$ is cancelled out, we obtain the following gap-expanding relationship:

$$\mathbb{E}_{\boldsymbol{a}\sim\pi_\beta(\boldsymbol{a}|\boldsymbol{s})}[\hat{Q}^{k+1}(\boldsymbol{s}, \boldsymbol{a})] - \mathbb{E}_{\pi_\beta(\boldsymbol{a}|\boldsymbol{s})}[Q^{k+1}(\boldsymbol{s}, \boldsymbol{a})] > \mathbb{E}_{\mu_k(\boldsymbol{a}|\boldsymbol{s})}[\hat{Q}^{k+1}(\boldsymbol{s}, \boldsymbol{a})] - \mathbb{E}_{\mu_k(\boldsymbol{a}|\boldsymbol{s})}[Q^{k+1}(\boldsymbol{s}, \boldsymbol{a})]$$

for, α_k satisfying,

$$\alpha_k > \max\left(\frac{(\pi_\beta(\boldsymbol{a}|\boldsymbol{s}) - \mu_k(\boldsymbol{a}|\boldsymbol{s}))^T \left[\mathcal{B}^{\pi^k}\left(\hat{Q}^k - Q^k\right)(\boldsymbol{s}, \cdot)\right]}{\hat{\Delta}^k}, 0\right),$$

thus proving the desired result.

To avoid the dependency on the true Q-value iterate, Q^k, we can upper-bound Q^k by $\frac{R_{\max}}{1-\gamma}$, and upper-bound $(\pi_\beta(\boldsymbol{a}|\boldsymbol{s}) - \mu_k(\boldsymbol{a}|\boldsymbol{s}))^T \mathcal{B}^{\pi^k} Q^k(\boldsymbol{s}, \cdot)$ by $D_{\text{TV}}(\pi_\beta, \mu_k) \cdot \frac{R_{\max}}{1-\gamma}$, and use this in the expression for α_k. While this bound may be loose, it still guarantees the gap-expanding property, and we indeed empirically show the existence of this property in practice in Appendix C.2.

To incorporate sampling error, we can follow a similar strategy as previous proofs: the worst case overestimation due to sampling error is given by $\frac{C_{r,T,\delta} R_{\max}}{1-\gamma}$. In this case, we note that, w.h.p.,

$$\left|\hat{\mathcal{B}}^{\pi^k}\left(\hat{Q}^k - Q^k\right) - \mathcal{B}^{\pi^k}\left(\hat{Q}^k - Q^k\right)\right| \leq \frac{2 \cdot C_{r,T,\delta} R_{\max}}{1-\gamma}.$$

Hence, the presence of sampling error adds $D_{\text{TV}}(\hat{\pi}_\beta, \mu_k) \cdot \frac{2 \cdot C_{r,T,\delta} R_{\max}}{1-\gamma}$ to the value of α_k, giving rise to the following, sufficient condition on α_k for the gap-expanding property:

$$\alpha_k > \max\left(\frac{(\pi_\beta(\boldsymbol{a}|\boldsymbol{s}) - \mu_k(\boldsymbol{a}|\boldsymbol{s}))^T \left[\mathcal{B}^{\pi^k}\left(\hat{Q}^k - Q^k\right)(\boldsymbol{s}, \cdot)\right]}{\hat{\Delta}^k} + D_{\text{TV}}(\hat{\pi}_\beta, \mu_k) \cdot \frac{2 \cdot C_{r,T,\delta} R_{\max}}{1-\gamma}, 0\right),$$

concluding the proof of this theorem.

C.4 Additional Theoretical Analysis

In this section, we present a theoretical analysis of additional properties of CQL. For ease of presentation, we state and prove theorems in Appendices C.4.1 and C.4.2 in the absence of sampling error, but as discussed extensively in Appendix C.3, we can extend each of these results by adding extra terms induced due to sampling error.

C.4.1 *CQL with Linear and Non-Linear Function Approximation*

Theorem C.4.1. *Assume that the Q-function is represented as a linear function of given state-action feature vectors $\boldsymbol{F}$, i.e., $Q(s,a) = w^T \boldsymbol{F}(s,a)$. Let $D = \text{diag}\left(d^{\pi_\beta}(\boldsymbol{s})\pi_\beta(\boldsymbol{a}|\boldsymbol{s})\right)$ denote the diagonal matrix with data density, and assume that $\boldsymbol{F}^T D \boldsymbol{F}$ is invertible. Then, the expected value of the policy under Q-value from Eqn 5.1.2 at iteration $k+1$, $\mathbb{E}_{d^{\pi_\beta}(\boldsymbol{a})}[\hat{V}^{k+1}(\boldsymbol{s})] =$*

$\mathbb{E}_{d^{\pi_\beta}(\boldsymbol{s}),\pi(\boldsymbol{a}|\boldsymbol{s})}[\hat{Q}^{k+1}(\boldsymbol{s},\boldsymbol{a})]$, *lower-bounds the corresponding tabular value,* $\mathbb{E}_{d^{\pi_\beta}(\boldsymbol{s})}[V^{k+1}(\boldsymbol{s})] = \mathbb{E}_{d^{\pi_\beta}(\boldsymbol{s}),\pi(\boldsymbol{a}|\boldsymbol{s})}[Q^{k+1}(\boldsymbol{s},\boldsymbol{a})]$, *if*

$$\alpha_k \geq \max\left(\frac{D^T\left[\boldsymbol{F}\left(\boldsymbol{F}^T D\boldsymbol{F}\right)^{-1}\boldsymbol{F}^T - I\right]\left((\mathcal{B}^\pi\hat{Q}^k)(\boldsymbol{s},\boldsymbol{a})\right)}{D^T\left[\boldsymbol{F}\left(\boldsymbol{F}^T D\boldsymbol{F}\right)^{-1}\boldsymbol{F}^T\right]\left(D\left[\frac{\pi(\boldsymbol{a}|\boldsymbol{s})-\pi_\beta(\boldsymbol{a}|\boldsymbol{s})}{\pi_\beta(\boldsymbol{a}|\boldsymbol{s})}\right]\right)}, 0\right).$$

The choice of α_k in Theorem C.4.1 intuitively amounts to compensating for overestimation in value induced if the true value function cannot be represented in the chosen linear function class (numerator), by the potential decrease in value due to the CQL regularizer (denominator). This implies that if the actual value function can be represented in the linear function class, such that the numerator can be made 0, then **any** $\alpha > 0$ is sufficient to obtain a lower bound. We now prove the theorem.

Proof. In order to extend the result of Theorem 5.1.2 to account for function approximation, we follow the similar recipe as before. We obtain the optimal solution to the optimization problem below in the family of linearly expressible Q-functions, i.e. $\mathcal{Q} := \{\boldsymbol{F}w | w \in \mathbb{R}^{dim(\boldsymbol{F})}\}$.

$$\alpha_k \cdot \left(\mathbb{E}_{d^{\pi_\beta}(\boldsymbol{s}),\mu(\boldsymbol{a}|\boldsymbol{s})}\left[Q(\boldsymbol{s},\boldsymbol{a})\right] - \mathbb{E}_{d^{\pi_\beta}(\boldsymbol{s}),\pi_\beta(\boldsymbol{a}|\boldsymbol{s})}\left[Q(\boldsymbol{s},\boldsymbol{a})\right]\right) + \frac{1}{2}\,\mathbb{E}_{\mathcal{D}}\left[\left((Q - \mathcal{B}^\pi\hat{Q}^k)(\boldsymbol{s},\boldsymbol{a})\right)^2\right].$$

By substituting $Q(\boldsymbol{s},\boldsymbol{a}) = w^T\boldsymbol{F}(\boldsymbol{s},\boldsymbol{a})$, and setting the derivative with respect to w to be 0, we obtain,

$$\alpha\sum_{\boldsymbol{s},\boldsymbol{a}} d^{\pi_\beta}(\boldsymbol{s})\cdot\left((\mu-\pi_\beta)(\boldsymbol{a}|\boldsymbol{s})\right)\boldsymbol{F}(\boldsymbol{s},\boldsymbol{a}) + \sum_{\boldsymbol{s},\boldsymbol{a}} d^{\pi_\beta}(\boldsymbol{s})\pi_\beta(\boldsymbol{a}|\boldsymbol{s})\left((Q-\mathcal{B}^\pi\hat{Q}^k)(\boldsymbol{s},\boldsymbol{a})\right)\boldsymbol{F}(\boldsymbol{s},\boldsymbol{a}) = 0.$$

By re-arranging terms, and converting it to vector notation, defining $D = \text{diag}(d^{\pi_\beta}(\boldsymbol{s})\pi_\beta(\boldsymbol{s}))$, and referring to the parameter w at the k-iteration as w^k we obtain:

$$\left(\boldsymbol{F}^T D\boldsymbol{F}\right)w^{k+1} = \underbrace{\boldsymbol{F}^T D\left(\mathcal{B}^\pi\hat{Q}^k\right)}_{\text{LSTD iterate}} - \underbrace{\alpha_k\boldsymbol{F}^T\text{diag}\left[d^{\pi_\beta}(\boldsymbol{s})\cdot(\mu(\boldsymbol{a}|\boldsymbol{s}) - \pi_\beta(\boldsymbol{a}|\boldsymbol{s}))\right]}_{\text{underestimation}}.$$

Now, our task is to show that the term labelled as "underestimation" is indeed negative in expectation under $\mu(\boldsymbol{a}|\boldsymbol{s})$ (This is analogous to our result in the tabular setting that shows underestimated values). In order to show this, we write out the expression for the value, under the linear weights w^{k+1} at state $\boldsymbol{s}$, after substituting $\mu = \pi$,

$$\hat{V}^{k+1}(\boldsymbol{s}) := \pi(\boldsymbol{a}|\boldsymbol{s})^T\boldsymbol{F}w^{k+1} \tag{C.4.1}$$

$$= \pi(\boldsymbol{a}|\boldsymbol{s})^T\underbrace{\boldsymbol{F}\left(\boldsymbol{F}^T D\boldsymbol{F}\right)^{-1}\boldsymbol{F}^T D\left(\mathcal{B}^\pi\hat{Q}^k\right)}_{\text{value under LSTD-Q [191]}} - \alpha_k\pi(\boldsymbol{a}|\boldsymbol{s})^T\boldsymbol{F}\left(\boldsymbol{F}^T D\boldsymbol{F}\right)^{-1}\boldsymbol{F}^T D\left[\frac{\pi(\boldsymbol{a}|\boldsymbol{s}) - \pi_\beta(\boldsymbol{a}|\boldsymbol{s})}{\pi_\beta(\boldsymbol{a}|\boldsymbol{s})}\right]. \tag{C.4.2}$$

Now, we need to reason about this additional penalty term. Defining, $P_{\boldsymbol{F}:=\boldsymbol{F}(\boldsymbol{F}^T D \boldsymbol{F})^{-1}\boldsymbol{F}^T D}$ as the projection matrix onto the subspace of features $\boldsymbol{F}$, we need to show that the product that appears as a penalty is positive: $\pi(\boldsymbol{a}|\boldsymbol{s})^T P_{\boldsymbol{F}} \left[\frac{\pi(\boldsymbol{a}|\boldsymbol{s}) - \pi_\beta(\boldsymbol{a}|\boldsymbol{s})}{\pi_\beta(\boldsymbol{a}|\boldsymbol{s})}\right] \geq 0$. In order to show this, we compute minimum value of this product optimizing over π. If the minimum value is 0, then we are done.

Let's define $f(\pi) = \pi(\boldsymbol{a}|\boldsymbol{s})^T P_{\boldsymbol{F}\left[\frac{\pi(\boldsymbol{a}|\boldsymbol{s}) - \pi_\beta(\boldsymbol{a}|\boldsymbol{s})}{\pi_\beta(\boldsymbol{a}|\boldsymbol{s})}\right]}$, our goal is to solve for $\min_\pi f(\pi)$. Setting the derivative of $f(\pi)$ with respect to π to be equal to 0, we obtain (including Lagrange multiplier η that guarantees $\sum_{\boldsymbol{a}} \pi(\boldsymbol{a}|\boldsymbol{s}) = 1$,

$$\left(P_{\boldsymbol{F}+P_{\boldsymbol{F}^T}}\right)\left[\frac{\pi(\boldsymbol{a}|\boldsymbol{s})}{\pi_\beta(\boldsymbol{a}|\boldsymbol{s})}\right] = P_{\boldsymbol{F}\vec{1}+\eta\vec{1}.}$$

By solving for η (using the condition that a density function sums to 1), we obtain that the minimum value of $f(\pi)$ occurs at a $\pi^*(\boldsymbol{a}|\boldsymbol{s})$, which satisfies the following condition,

$$\left(P_{\boldsymbol{F}+P_{\boldsymbol{F}^T}}\right)\left[\frac{\pi^*(\boldsymbol{a}|\boldsymbol{s})}{\pi_\beta(\boldsymbol{a}|\boldsymbol{s})}\right] = \left(P_{\boldsymbol{F}+P_{\boldsymbol{F}^T}}\right)\vec{1}.$$

Using this relation to compute f, we obtain, $f(\pi^*) = 0$, indicating that the minimum value of 0 occurs when the projected density ratio matches under the matrix $(P_{\boldsymbol{F}+P_{\boldsymbol{F}^T}})$ is equal to the projection of a vector of ones, $\vec{1}$. Thus,

$$\forall\ \pi(\boldsymbol{a}|\boldsymbol{s}),\ f(\pi) = \pi(\boldsymbol{a}|\boldsymbol{s})^T P_{\boldsymbol{F}}\left[\frac{\pi(\boldsymbol{a}|\boldsymbol{s}) - \pi_\beta(\boldsymbol{a}|\boldsymbol{s})}{\pi_\beta(\boldsymbol{a}|\boldsymbol{s})}\right] \geq 0.$$

This means that: $\forall\ \boldsymbol{s}, \hat{V}^{k+1}(\boldsymbol{s}) \leq \hat{V}^{k+1}_{\text{LSTD-Q}}(\boldsymbol{s})$ given identical previous $\hat{Q}^k$ values. This result indicates, that if $\alpha_k \geq 0$, the resulting CQL value estimate with linear function approximation is guaranteed to lower-bound the value estimate obtained from a least-squares temporal difference Q-learning algorithm (which only minimizes Bellman error assuming a linear Q-function parameterization), such as LSTD-Q [191], since at each iteration, CQL induces a lower-bound with respect to the previous value iterate, whereas this underestimation is absent in LSTD-Q. Also note that we can also apply an inductive argument.

So far, we have only shown that the learned value iterate, $\hat{V}^{k+1}(\boldsymbol{s})$ lower-bounds the value iterate obtained from LSTD-Q, $\forall\ \boldsymbol{s}, \hat{V}^{k+1}(\boldsymbol{s}) \leq \hat{V}^{k+1}_{\text{LSTD-Q}}(\boldsymbol{s})$. But, our final aim is to prove a stronger result, that the learned value iterate, $\hat{V}^{k+1}$, lower bounds the exact tabular value function iterate, V^{k+1}, at each iteration. The reason why our current result does not guarantee this is because function approximation may induce overestimation error in the linear approximation of the Q-function.

In order to account for this change, we make a simple change: we choose α_k such that the resulting penalty nullifes the effect of any over-estimation caused due to the inability to fit the true value function iterate in the linear function class parameterized by $\boldsymbol{F}$. Formally, this means:

$$\begin{aligned}
\mathbb{E}_{d^{\pi_\beta}(\boldsymbol{s})}\left[\hat{V}^{k+1}(\boldsymbol{s})\right] &\leq \mathbb{E}_{d^{\pi_\beta}(\boldsymbol{s})}\left[\hat{V}^{k+1}_{\text{LSTD-Q}}(\boldsymbol{s})\right] - \alpha_k \mathbb{E}_{d^{\pi_\beta}(\boldsymbol{s})}[f(\pi(\boldsymbol{a}|\boldsymbol{s}))] \\
\leq \mathbb{E}_{d^{\pi_\beta}(\boldsymbol{s})}\left[V^{k+1}(\boldsymbol{s})\right] &\underbrace{- \mathbb{E}_{d^{\pi_\beta}(\boldsymbol{s})}\left[\hat{V}^{k+1}_{\text{LSTD-Q}}(\boldsymbol{s}) - V^{k+1}(\boldsymbol{s})\right] - \alpha_k \mathbb{E}_{d^{\pi_\beta}(\boldsymbol{s})}[f(\pi(\boldsymbol{a}|\boldsymbol{s}))]}_{\text{choose } \alpha_k \text{ to make this negative}} \\
&\leq \mathbb{E}_{d^{\pi_\beta}(\boldsymbol{s})}\left[V^{k+1}(\boldsymbol{s})\right]
\end{aligned}$$

And the choice of α_k in that case is given by:

$$\begin{aligned}
\alpha_k &\geq \max\left(\frac{\mathbb{E}_{d^{\pi_\beta}(\boldsymbol{s})}\left[\hat{V}^{k+1}_{\text{LSTD-Q}}(\boldsymbol{s}) - V^{k+1}(\boldsymbol{s})\right]}{\mathbb{E}_{d^{\pi_\beta}(\boldsymbol{s})}[f(\pi(\boldsymbol{a}|\boldsymbol{s}))]}, 0\right) \\
&\geq \max\left(\frac{D^T\left[\boldsymbol{F}\left(\boldsymbol{F}^T D \boldsymbol{F}\right)^{-1}\boldsymbol{F}^T - I\right]\left((\mathcal{B}^\pi \hat{Q}^k)(\boldsymbol{s},\boldsymbol{a})\right)}{D^T\left[\boldsymbol{F}\left(\boldsymbol{F}^T D \boldsymbol{F}\right)^{-1}\boldsymbol{F}^T\right]\left(D\left[\frac{\pi(\boldsymbol{a}|\boldsymbol{s}) - \pi_\beta(\boldsymbol{a}|\boldsymbol{s})}{\pi_\beta(\boldsymbol{a}|\boldsymbol{s})}\right]\right)}, 0\right).
\end{aligned}$$

Finally, we note that since this choice of α_k induces under-estimation in the next iterate, $\hat{V}^{k+1}$ with respect to the previous iterate, $\hat{V}^k$, for all $k \in \mathbb{N}$, by induction, we can claim that this choice of $\alpha_1, \cdots, \alpha_k$ is sufficient to make $\hat{V}^{k+1}$ lower-bound the tabular, exact value-function iterate. V^{k+1}, for all k, thus completing our proof. □

We can generalize Theorem C.4.1 to non-linear function approximation, such as neural networks, under the standard NTK framework [145], assuming that each iteration k is performed by a single step of gradient descent on Equation 5.1.2, rather than a complete minimization of this objective. As we show in Theorem C.4.2, CQL learns lower bounds in this case for an appropriate choice of α_k.

Theorem C.4.2 (Extension to non-linear function approximation)**.** *Assume that the Q-function is represented by a general non-linear function approximator parameterized by θ, $Q_\theta(\boldsymbol{s},\boldsymbol{a})$. let $D = diag(d^{\pi_\beta}(\boldsymbol{s})\pi_\beta(\boldsymbol{a}|\boldsymbol{s}))$ denote the matrix with the data density on the diagonal, and assume that $\nabla_\theta Q_\theta^T D \nabla_\theta Q_\theta$ is invertible. Then, the expected value of the policy under the Q-function obtained by taking a gradient step on Equation 5.1.2, at iteration $k+1$ lower-bounds the corresponding tabular function iterate if:*

Proof. Our proof strategy is to reduce the non-linear optimization problem into a linear one, with features $\boldsymbol{F}$ (in Theorem C.4.1) replaced with features given by the gradient of the current Q-function $\hat{Q}^k_\theta$ with respect to parameters θ, i.e. $\nabla_\theta \hat{Q}^k$. To see, this we start by

writing down the expression for $\hat{Q}_\theta^{k+1}$ obtained via one step of gradient descent with step size η, on the objective in Equation 5.1.2.

$$\theta^{k+1} = \theta^k - \eta\alpha_k \left(\mathbb{E}_{d^{\pi_\beta}(\boldsymbol{s}),\mu(\boldsymbol{a}|\boldsymbol{s})} \left[\nabla_\theta \hat{Q}^k(\boldsymbol{s},\boldsymbol{a}) \right] - \mathbb{E}_{d^{\pi_\beta}(\boldsymbol{s}),\pi_\beta(\boldsymbol{a}|\boldsymbol{s})} \left[\nabla_\theta \hat{Q}^k(\boldsymbol{s},\boldsymbol{a}) \right] \right) - \eta \mathbb{E}_{d^{\pi_\beta}(\boldsymbol{s}),\pi_\beta(\boldsymbol{a}|\boldsymbol{s})} \left[\left(\hat{Q}^k - \mathcal{B}^\pi \hat{Q}^k \right)(\boldsymbol{s},\boldsymbol{a}) \cdot \nabla_\theta \hat{Q}^k(\boldsymbol{s},\boldsymbol{a}) \right].$$

Using the above equation and making an approximation linearization assumption on the non-linear Q-function, for small learning rates $\eta << 1$, as has been commonly used by prior works on the neural tangent kernel (NTK) in deep learning theory [145] in order to explain neural network learning dynamics in the infinite-width limit, we can write out the expression for the next Q-function iterate, $\hat{Q}_\theta^{k+1}$ in terms of $\hat{Q}_\theta^k$ as [4, 145]:

$$\begin{aligned}
\hat{Q}_\theta^{k+1}(\boldsymbol{s},\boldsymbol{a}) &\approx \hat{Q}_\theta^k(\boldsymbol{s},\boldsymbol{a}) + \left(\theta^{k+1} - \theta^k\right)^T \nabla_\theta \hat{Q}_\theta^k(\boldsymbol{s},\boldsymbol{a}) \quad \text{(under NTK assumptions)} \\
&= \hat{Q}_\theta^k(\boldsymbol{s},\boldsymbol{a}) - \eta\alpha_k \mathbb{E}_{d^{\pi_\beta}(\boldsymbol{s}'),\mu(\boldsymbol{a}'|\boldsymbol{s}')} \left[\nabla_\theta \hat{Q}^k(\boldsymbol{s}',\boldsymbol{a}')^T \nabla_\theta \hat{Q}^k(\boldsymbol{s},\boldsymbol{a}) \right] \\
&\quad + \eta\alpha_k \mathbb{E}_{d^{\pi_\beta}(\boldsymbol{s}'),\pi_\beta(\boldsymbol{a}'|\boldsymbol{s}')} \left[\nabla_\theta \hat{Q}^k(\boldsymbol{s}',\boldsymbol{a}')^T \nabla_\theta \hat{Q}^k(\boldsymbol{s},\boldsymbol{a}) \right] \\
&\quad - \eta \mathbb{E}_{d^{\pi_\beta}(\boldsymbol{s}'),\pi_\beta(\boldsymbol{a}'|\boldsymbol{s}')} \left[\left(\hat{Q}^k - \mathcal{B}^\pi \hat{Q}^k \right)(\boldsymbol{s}',\boldsymbol{a}') \cdot \nabla_\theta \hat{Q}^k(\boldsymbol{s}',\boldsymbol{a}')^T \nabla_\theta \hat{Q}^k(\boldsymbol{s},\boldsymbol{a}) \right].
\end{aligned}$$

To simplify presentation, we convert into matrix notation, where we define the $|\mathcal{S}||\mathcal{A}| \times |\mathcal{S}||\mathcal{A}|$ matrix, $\boldsymbol{M}^k = \left(\nabla_\theta \hat{Q}^k\right)^T \nabla_\theta \hat{Q}^k$, as the neural tangent kernel matrix of the Q-function at iteration k. Then, the vectorized $\hat{Q}^{k+1}$ (with $\mu = \pi$) is given by,

$$\hat{Q}^{k+1} = \hat{Q}^k - \eta\alpha_k \boldsymbol{M}^k D \left[\frac{\pi(\boldsymbol{a}|\boldsymbol{s}) - \pi_\beta(\boldsymbol{a}|\boldsymbol{s})}{\pi_\beta(\boldsymbol{a}|\boldsymbol{s})} \right] + \eta \boldsymbol{M}^k D \left(\mathcal{B}^\pi \hat{Q}^k - \hat{Q}^k \right).$$

Finally, the value of the policy is given by:

$$\hat{V}^{k+1} := \underbrace{\pi(\boldsymbol{a}|\boldsymbol{s})^T \hat{Q}^k(\boldsymbol{s},\boldsymbol{a}) + \eta\pi(\boldsymbol{a}|\boldsymbol{s}) \boldsymbol{M}^k D \left(\mathcal{B}^\pi \hat{Q}^k - \hat{Q}^k \right)}_{\text{(a) unpenalized value}} \tag{C.4.3}$$

$$\underbrace{- \eta\alpha_k \pi(\boldsymbol{a}|\boldsymbol{s})^T \boldsymbol{M}^k D \left[\frac{\pi(\boldsymbol{a}|\boldsymbol{s}) - \pi_\beta(\boldsymbol{a}|\boldsymbol{s})}{\pi_\beta(\boldsymbol{a}|\boldsymbol{s})} \right]}_{\text{(b) penalty}}. \tag{C.4.4}$$

Term marked (b) in the above equation is similar to the penalty term shown in Equation C.4.2, and by performing a similar analysis, we can show that (b) ≥ 0. Again similar to how $\hat{V}^{k+1}_{\text{LSTD-Q}}$ appeared in Equation C.4.2, we observe that here we obtain the value function corresponding to a regular gradient-step on the Bellman error objective. □

Again similar to before, term (a) can introduce overestimation relative to the tabular counterpart, starting at $\hat{Q}^k$: $Q^{k+1} = \hat{Q}^k - \eta \left(\mathcal{B}^\pi \hat{Q}^k - \hat{Q}^k\right)$, and we can choose α_k to compensate for this potential increase as in the proof of Theorem C.4.1. As the last step, we can recurse this argument to obtain our final result, for underestimation.

C.4.2 *Choice of Distribution to Maximize Expected Q-Value in Equation 5.1.2*

In Section 5.1.2, we introduced a term that maximizes Q-values under the dataset $d^{\pi_\beta}(\boldsymbol{s})\pi_\beta(\boldsymbol{a}|\boldsymbol{s})$ distribution when modifying Equation 5.1.1 to Equation 5.1.2. Theorem 5.1.2 indicates the "sufficiency" of maximizing Q-values under the dataset distribution – this guarantees a lower-bound on value. We now investigate the neccessity of this assumption: We ask the formal question: **For which other choices of $\nu(\boldsymbol{a}|\boldsymbol{s})$ for the maximization term, is the value of the policy under the learned Q-value, $\hat{Q}^{k+1}_\nu$ guaranteed to be a lower bound on the actual value of the policy?**

To recap and define notation, we restate the objective from Equation 5.1.1 below.

$$\hat{Q}^{k+1} \leftarrow \arg\min_Q \; \alpha \, \mathbb{E}_{\boldsymbol{s}\sim d^{\pi_\beta}(\boldsymbol{s}),\boldsymbol{a}\sim\mu(\boldsymbol{a}|\boldsymbol{s})}\left[Q(\boldsymbol{s},\boldsymbol{a})\right] + \frac{1}{2}\,\mathbb{E}_{\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}'\sim\mathcal{D}}\left[\left(Q(\boldsymbol{s},\boldsymbol{a}) - \mathcal{B}^\pi\hat{Q}^k(\boldsymbol{s},\boldsymbol{a})\right)^2\right]. \quad \text{(C.4.5)}$$

We define a general family of objectives from Equation 5.1.2, parameterized by a distribution ν which is chosen to maximize Q-values as shown below (CQL is a special case, with $\nu(\boldsymbol{a}|\boldsymbol{s}) = \pi_\beta(\boldsymbol{a}|\boldsymbol{s})$):

$$\hat{Q}^{k+1}_\nu \leftarrow \arg\min_Q \; \alpha \cdot \left(\mathbb{E}_{\boldsymbol{s}\sim d^{\pi_\beta}(\boldsymbol{s}),\boldsymbol{a}\sim\mu(\boldsymbol{a}|\boldsymbol{s})}\left[Q(\boldsymbol{s},\boldsymbol{a})\right] - \mathbb{E}_{\boldsymbol{s}\sim d^{\pi_\beta}(\boldsymbol{s}),\boldsymbol{a}\sim\nu(\boldsymbol{a}|\boldsymbol{s})}\left[Q(\boldsymbol{s},\boldsymbol{a})\right]\right)$$
$$+\frac{1}{2}\,\mathbb{E}_{\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}'\sim\mathcal{D}}\left[\left(Q(\boldsymbol{s},\boldsymbol{a}) - \mathcal{B}^\pi\hat{Q}^k(\boldsymbol{s},\boldsymbol{a})\right)^2\right]. \quad (\nu\text{-CQL}) \quad \text{(C.4.6)}$$

In order to answer our question, we prove the following result:

Theorem C.4.3 (Necessity of maximizing Q-values under $\pi_\beta(\boldsymbol{a}|\boldsymbol{s})$.)**.** *For any policy $\pi(\boldsymbol{a}|\boldsymbol{s})$, any $\alpha > 0$, and for all $k > 0$, the value of the policy., $\hat{V}^{k+1}_\nu$ under Q-function iterates from $\nu-CQL$, $\hat{Q}^{k+1}_\nu(\boldsymbol{s},\boldsymbol{a})$ is guaranteed to be a lower bound on the exact value iterate, $\hat{V}^{k+1}$, only if $\nu(\boldsymbol{a}|\boldsymbol{s}) = \pi_\beta(\boldsymbol{a}|\boldsymbol{s})$.*

Proof. We start by noting the parametric form of the resulting tabular Q-value iterate:

$$\hat{Q}^{k+1}_\nu(\boldsymbol{s},\boldsymbol{a}) = \mathcal{B}^\pi\hat{Q}^k_\nu(\boldsymbol{s},\boldsymbol{a}) - \alpha_k\frac{\mu(\boldsymbol{a}|\boldsymbol{s}) - \nu(\boldsymbol{a}|\boldsymbol{s})}{\pi_\beta(\boldsymbol{a}|\boldsymbol{s})}. \quad \text{(C.4.7)}$$

The value of the policy under this Q-value iterate, when distribution μ is chosen to be the target policy $\pi(\boldsymbol{a}|\boldsymbol{s})$ i.e. $\mu(\boldsymbol{a}|\boldsymbol{s}) = \pi(\boldsymbol{a}|\boldsymbol{s})$ is given by:

$$\hat{V}^{k+1}_\nu(\boldsymbol{s}) := \mathbb{E}_{\boldsymbol{a}\sim\pi(\boldsymbol{a}|\boldsymbol{s})}\left[\hat{Q}^{k+1}_\nu(\boldsymbol{s},\boldsymbol{a})\right] = \mathbb{E}_{\boldsymbol{a}\sim\pi(\boldsymbol{a}|\boldsymbol{s})}\left[\mathcal{B}^\pi\hat{Q}^k_\nu(\boldsymbol{s},\boldsymbol{a})\right] - \alpha_k\,\pi(\boldsymbol{a}|\boldsymbol{s})^T\left(\frac{\pi(\boldsymbol{a}|\boldsymbol{s}) - \nu(\boldsymbol{a}|\boldsymbol{s})}{\pi_\beta(\boldsymbol{a}|\boldsymbol{s})}\right). \quad \text{(C.4.8)}$$

We are interested in conditions on $\nu(\boldsymbol{a}|\boldsymbol{s})$ such that the penalty term in the above equation is positive. It is clear that choosing $\nu(\boldsymbol{a}|\boldsymbol{s}) = \pi_\beta(\boldsymbol{a}|\boldsymbol{s})$ returns a policy that satisfies the requirement, as shown in the proof for Theorem 5.1.2. In order to obtain other choices of $\nu(\boldsymbol{a}|\boldsymbol{s})$ that guarantees a lower bound for all possible choices of $\pi(\boldsymbol{a}|\boldsymbol{s})$, we solve the following concave-convex maxmin optimization problem, that computes a $\nu(\boldsymbol{a}|\boldsymbol{s})$ for which a lower-bound is guaranteed for *all* choices of $\mu(\boldsymbol{a}|\boldsymbol{s})$:

$$\begin{aligned}\max_{\nu(\boldsymbol{a}|\boldsymbol{s})} \min_{\pi(\boldsymbol{a}|\boldsymbol{s})} \quad & \sum_{\boldsymbol{a}} \pi(\boldsymbol{a}|\boldsymbol{s}) \cdot \left(\frac{\pi(\boldsymbol{a}|\boldsymbol{s}) - \nu(\boldsymbol{a}|\boldsymbol{s})}{\pi_\beta(\boldsymbol{a}|\boldsymbol{s})} \right) \\ \text{s.t.} \quad & \sum_{\boldsymbol{a}} \pi(\boldsymbol{a}|\boldsymbol{s}) = 1, \ \sum_{\boldsymbol{a}} \nu(\boldsymbol{a}|\boldsymbol{s}) = 1, \ \nu(\boldsymbol{a}|\boldsymbol{s}) \geq 0, \ \pi(\boldsymbol{a}|\boldsymbol{s}) \geq 0.\end{aligned}$$

We first solve the inner minimization over $\pi(\boldsymbol{a}|\boldsymbol{s})$ for a fixed $\nu(\boldsymbol{a}|\boldsymbol{s})$, by writing out the Lagrangian and setting the gradient of the Lagrangian to be 0, we obtain:

$$\forall \boldsymbol{a}, \ \ 2 \cdot \frac{\pi^*(\boldsymbol{a}|\boldsymbol{s})}{\pi_\beta(\boldsymbol{a}|\boldsymbol{s})} - \frac{\nu(\boldsymbol{a}|\boldsymbol{s})}{\pi_\beta(\boldsymbol{a}|\boldsymbol{s})} - \zeta(\boldsymbol{a}|\boldsymbol{s}) + \eta = 0,$$

where $\zeta(\boldsymbol{a}|\boldsymbol{s})$ is the Lagrange dual variable for the positivity constraints on $\pi(\boldsymbol{a}|\boldsymbol{s})$, and η is the Lagrange dual variable for the normalization constraint on π. If $\pi(\boldsymbol{a}|\boldsymbol{s})$ is full support (for example, when it is chosen to be a Boltzmann policy), KKT conditions imply that, $\zeta(\boldsymbol{a}|\boldsymbol{s}) = 0$, and computing η by summing up over actions, $\boldsymbol{a}$, the optimal choice of π for the inner minimization is given by:

$$\pi^*(\boldsymbol{a}|\boldsymbol{s}) = \frac{1}{2}\nu(\boldsymbol{a}|\boldsymbol{s}) + \frac{1}{2}\pi_\beta(\boldsymbol{a}|\boldsymbol{s}). \tag{C.4.9}$$

Now, plugging Equation C.4.9 in the original optimization problem, we obtain the following optimization over only $\nu(\boldsymbol{a}|\boldsymbol{s})$:

$$\max_{\nu(\boldsymbol{a}|\boldsymbol{s})} \sum_{\boldsymbol{a}} \pi_\beta(\boldsymbol{a}|\boldsymbol{s}) \cdot \left(\frac{1}{2} - \frac{\nu(\boldsymbol{a}|\boldsymbol{s})}{2\pi_\beta(\boldsymbol{a}|\boldsymbol{s})} \right) \cdot \left(\frac{1}{2} + \frac{\nu(\boldsymbol{a}|\boldsymbol{s})}{2\pi_\beta(\boldsymbol{a}|\boldsymbol{s})} \right) \quad \text{s.t.} \ \sum_{\boldsymbol{a}} \nu(\boldsymbol{a}|\boldsymbol{s}) = 1, \ \nu(\boldsymbol{a}|\boldsymbol{s}) \geq 0. \tag{C.4.10}$$

Solving this optimization, we find that the optimal distribution, $\nu(\boldsymbol{a}|\boldsymbol{s})$ is equal to $\pi_\beta(\boldsymbol{a}|\boldsymbol{s})$. and the optimal value of penalty, which is also the objective for the problem above is equal to 0. Since we are maximizing over ν, this indicates for other choices of $\nu \neq \pi_\beta$, we can find a π so that the penalty is negative, and hence a lower-bound is not guaranteed. Therefore, we find that with a worst case choice of $\pi(\boldsymbol{a}|\boldsymbol{s})$, a lower bound can only be guaranteed only if $\nu(\boldsymbol{a}|\boldsymbol{s}) = \pi_\beta(\boldsymbol{a}|\boldsymbol{s})$. This justifies the necessity of $\pi_\beta(\boldsymbol{a}|\boldsymbol{s})$ for maximizing Q-values in Equation 5.1.2. The above analysis doesn't take into account the effect of function approximation or sampling error. We can, however, generalize this result to those settings, by following a similar strategy of appropriately choosing α_k, as previously utilized in Theorem C.4.1. □

C.4.3 *CQL with Empirical Dataset Distributions*

The results in Sections 5.1.2 and 5.1.3 account for sampling error due to the finite size of the dataset $\mathcal{D}$. In our practical implementation as well, we optimize a sample-based version of Equation 5.1.2, as shown below:

$$\hat{Q}^{k+1} \leftarrow \arg\min_{Q} \; \alpha \cdot \left(\sum_{s \in \mathcal{D}} \mathbb{E}_{a \sim \mu(a|s)} \left[Q(s,a) \right] - \sum_{s \in \mathcal{D}} \mathbb{E}_{a \sim \pi_\beta(a|s)} \left[Q(s,a) \right] \right) + \frac{1}{2|\mathcal{D}|} \sum_{s,a,s' \in \mathcal{D}} \left[\left(Q(s,a) - \hat{\mathcal{B}}^\pi \hat{Q}^k(s,a) \right)^2 \right], \quad \text{(C.4.11)}$$

where $\hat{\mathcal{B}}^\pi$ denotes the "empirical" Bellman operator computed using samples in $\mathcal{D}$ as follows:

$$\forall s, a \in \mathcal{D}, \quad \left(\hat{\mathcal{B}}^\pi \hat{Q}^k \right)(s,a) = r + \gamma \sum_{s'} \hat{T}(s'|s,a) \mathbb{E}_{a' \sim \pi(a'|s')} \left[\hat{Q}^k(s',a') \right], \quad \text{(C.4.12)}$$

where r is the empirical average reward obtained in the dataset when executing an action a at state s, i.e. $r = \frac{1}{|\mathcal{D}(s,a)|} \sum_{s_i, a_i in \mathcal{D}} \mathbf{1}_{s_i = s, a_i = a} \cdot r(s,a)$, and $\hat{T}(s'|s,a)$ is the empirical transition matrix. Note that expectation under $\pi(a|s)$ can be computed exactly, since it does not depend on the dataset. The empirical Bellman operator can take higher values as compared to the actual Bellman operator, $\mathcal{B}^\pi$, for instance, in an MDP with stochastic dynamics, where $\mathcal{D}$ may not contain transitions to all possible next-states s' that can be reached by executing action a at state s, and only contains an optimistic transition.

We next show how the CQL lower bound result (Theorem 5.1.2) can be modified to guarantee a lower bound even in this presence of sampling error. To note this, following prior work [146, 253], we assume concentration properties of the reward function and the transition dynamics:

Assumption C.4.4. *$\forall\, s, a \in \mathcal{D}$, the following relationships hold with high probability, $\geq 1 - \delta$*

$$|r - r(s,a)| \leq \frac{C_{r,\delta}}{\sqrt{|\mathcal{D}(s,a)|}}, \quad ||\hat{T}(s'|s,a) - T(s'|s,a)||_1 \leq \frac{C_{T,\delta}}{\sqrt{|\mathcal{D}(s,a)|}}.$$

Under this assumption, the difference between the empirical Bellman operator and the actual Bellman operator can be bounded:

$$\begin{aligned} \left| \left(\hat{\mathcal{B}}^\pi \hat{Q}^k \right) - \left(\mathcal{B}^\pi \hat{Q}^k \right) \right| &= \left| (r - r(s,a)) + \gamma \sum_{s'} \left(\hat{T}(s'|s,a) - T(s'|s,a) \right) \mathbb{E}_{\pi(a'|s')} \left[\hat{Q}^k(s',a') \right] \right| \\ &\leq |r - r(s,a)| + \gamma \left| \sum_{s'} \left(\hat{T}(s'|s,a) - T(s'|s,a) \right) \mathbb{E}_{\pi(a'|s')} \left[\hat{Q}^k(s',a') \right] \right| \\ &\leq \frac{C_{r,\delta} + \gamma C_{T,\delta} 2 R_{\max} / (1-\gamma)}{\sqrt{|\mathcal{D}(s,a)|}}. \end{aligned}$$

This gives us an expression to bound the overestimation due to sampling error, as a function of a constant, $C_{r,T,\delta}$ that can be expressed as a function of $C_{r,\delta}$ and $C_{T,\delta}$, and depends on δ via a $\sqrt{\log(1/\delta)}$ dependency. This is similar to how prior works have bounded the sampling error due to an empirical Bellman operator [253, 146].

C.4.4 *Safe Policy Improvement Guarantee for CQL*

In this section, we prove a safe policy improvement guarantee for CQL (and this analysis is also applicable in general to policy constraint methods with appropriate choices of constraints as we will show). We define the empirical MDP, $\hat{M}$ as the MDP formed by the transitions in the replay buffer, $\hat{M} = \{\boldsymbol{s}, \boldsymbol{a}, r, \boldsymbol{s}' \in \mathcal{D}\}$, and let $J(\pi, \hat{M})$ denote the return of a policy $\pi(\boldsymbol{a}|\boldsymbol{s})$ in MDP $\hat{M}$. Our goal is to show that $J(\pi, M) \geq J(\hat{\pi}_\beta, M) - \varepsilon$, with high probability, where ε is a small constant. We start by proving that CQL optimizes a penalized RL objective in the empirical MDP, $\hat{M}$.

Lemma C.4.5. *Let $\hat{Q}^\pi$ be the fixed point of Eq. 5.1.2, then $\pi^*(\boldsymbol{a}|\boldsymbol{s}) := \arg\max_\pi \mathbb{E}_{\boldsymbol{s}\sim\rho(\boldsymbol{s})}[\hat{V}^\pi(\boldsymbol{s})]$ is equivalently obtained by solving:*

$$\pi^*(\boldsymbol{a}|\boldsymbol{s}) \leftarrow \arg\max_\pi \; J(\pi, \hat{M}) - \alpha \frac{1}{1-\gamma} \mathbb{E}_{\boldsymbol{s}\sim d^\pi_{\hat{M}}(\boldsymbol{s})} \left[D_{CQL}(\pi, \hat{\pi}_\beta)(\boldsymbol{s}) \right], \tag{C.4.13}$$

where $D_{CQL}(\pi, \hat{\pi}_\beta)(\boldsymbol{s}) := \sum_{\boldsymbol{a}} \pi(\boldsymbol{a}|\boldsymbol{s}) \cdot \left(\frac{\pi(\boldsymbol{a}|\boldsymbol{s})}{\hat{\pi}_\beta(\boldsymbol{a}|\boldsymbol{s})} - 1 \right)$.

Proof. $\hat{Q}^\pi$ is obtained by solving a recursive Bellman fixed point equation in the empirical MDP $\hat{M}$, with an altered reward, $r(\boldsymbol{s}, \boldsymbol{a}) - \alpha \left[\frac{\pi(\boldsymbol{a}|\boldsymbol{s})}{\hat{\pi}_\beta(\boldsymbol{a}|\boldsymbol{s})} - 1 \right]$, hence the optimal policy $\pi^*(\boldsymbol{a}|\boldsymbol{s})$ obtained by optimizing the value under the CQL Q-function equivalently is characterized via Equation C.4.13. □

Now, our goal is to relate the performance of $\pi^*(\boldsymbol{a}|\boldsymbol{s})$ in the *actual* MDP, M, to the performance of the behavior policy, $\pi_\beta(\boldsymbol{a}|\boldsymbol{s})$. To this end, we prove the following theorem:

Theorem C.4.6. *Let $\pi^*(\boldsymbol{a}|\boldsymbol{s})$ be the policy obtained by optimizing Equation C.4.13. Then the performance of $\pi^*(\boldsymbol{a}|\boldsymbol{s})$ in the actual MDP M satisfies,*

$$J(\pi^*, M) \geq J(\hat{\pi}_\beta, M) \tag{C.4.14}$$

$$- 2 \left(\frac{C_{r,\delta}}{1-\gamma} + \frac{\gamma R_{\max} C_{T,\delta}}{(1-\gamma)^2} \right) \mathbb{E}_{\boldsymbol{s}\sim d^{\pi^*}_{\hat{M}}(\boldsymbol{s})} \left[\frac{\sqrt{|\mathcal{A}|}}{\sqrt{|\mathcal{D}(\boldsymbol{s})|}} \sqrt{D_{CQL}(\pi^*, \hat{\pi}_\beta)(\boldsymbol{s}) + 1} \right] \tag{C.4.15}$$

$$+ \alpha \frac{1}{1-\gamma} \mathbb{E}_{\boldsymbol{s}\sim d^{\pi^*}_{\hat{M}}(\boldsymbol{s})} \left[D_{CQL}(\pi^*, \hat{\pi}_\beta)(\boldsymbol{s}) \right]. \tag{C.4.16}$$

Proof. The proof for this statement is divided into two parts. The first part involves relating the return of $\pi^*(\boldsymbol{a}|\boldsymbol{s})$ in MDP $\hat{M}$ with the return of $\hat{\pi}_\beta$ in MDP $\hat{M}$. Since, $\pi^*(\boldsymbol{a}|\boldsymbol{s})$ optimizes Equation C.4.13, we can relate $J(\pi^*, \hat{M})$ and $J(\hat{\pi}_\beta, \hat{M})$ as:

$$J(\pi^*, \hat{M}) - \alpha \mathbb{E}_{\boldsymbol{s} \sim d^\pi_{\hat{M}}(\boldsymbol{s})} \left[D_{\text{CQL}}(\pi^*, \pi_\beta)(\boldsymbol{s}) \right] \geq J(\hat{\pi}_\beta, \hat{M}) - 0 = J(\hat{\pi}_\beta, \hat{M}).$$

The next step involves using concentration inequalities to upper and lower bound $J(\pi^*, \hat{M})$ and $J(\pi^*, M)$ and the corresponding difference for the behavior policy. In order to do so, we prove the following lemma, that relates $J(\pi, M)$ and $J(\pi, \hat{M})$ for an arbitrary policy π. We use this lemma to then obtain the proof for the above theorem. □

Lemma C.4.7. *For any MDP M, an empirical MDP $\hat{M}$ generated by sampling actions according to the behavior policy $\hat{\pi}_\beta(\boldsymbol{a}|\boldsymbol{s})$ and a given policy π,*

$$\left| J(\pi, \hat{M}) - J(\pi, M) \right| \leq \left(\frac{C_{r,\delta}}{1-\gamma} + \frac{\gamma R_{\max} C_{T,\delta}}{(1-\gamma)^2} \right) \mathbb{E}_{\boldsymbol{s} \sim d^\pi_{\hat{M}}(\boldsymbol{s})} \left[\frac{\sqrt{|\mathcal{A}|}}{|\mathcal{D}(\boldsymbol{s})|} \sqrt{D_{CQL}(\pi, \hat{\pi}_\beta)(\boldsymbol{s}) + 1} \right].$$

Proof. To prove this, we first use the triangle inequality to clearly separate reward and transition dynamics contributions in the expected return.

$$\left| J(\pi, \hat{M}) - J(\pi, M) \right| = \frac{1}{1-\gamma} \left| \sum_{\boldsymbol{s},\boldsymbol{a}} d^\pi_{\hat{M}}(\boldsymbol{s}) \pi(\boldsymbol{a}|\boldsymbol{s}) r_{\hat{M}}(\boldsymbol{s},\boldsymbol{a}) - \sum_{\boldsymbol{s},\boldsymbol{a}} d^\pi_M(\boldsymbol{s}) \pi(\boldsymbol{a}|\boldsymbol{s}) r_M(\boldsymbol{s},\boldsymbol{a}) \right| \tag{C.4.17}$$

$$\leq \frac{1}{1-\gamma} \left| \sum_{\boldsymbol{s},\boldsymbol{a}} d^\pi_{\hat{M}}(\boldsymbol{s}) \underbrace{\left[\pi(\boldsymbol{a}|\boldsymbol{s}) (r_{\hat{M}}(\boldsymbol{s},\boldsymbol{a}) - r_M(\boldsymbol{s},\boldsymbol{a}) \right]}_{:=\Delta_1(\boldsymbol{s})} \right| \tag{C.4.18}$$

$$+ \frac{1}{1-\gamma} \left| \sum_{\boldsymbol{s},\boldsymbol{a}} \left(d^\pi_{\hat{M}}(\boldsymbol{s}) - d^\pi_M(\boldsymbol{s}) \right) \pi(\boldsymbol{a}|\boldsymbol{s}) r_M(\boldsymbol{s},\boldsymbol{a}) \right| \tag{C.4.19}$$

We first use concentration inequalities to upper bound $\Delta_1(\boldsymbol{s})$. Note that under concentration assumptions, and by also using the fact that $\mathbb{E}[\Delta_1(\boldsymbol{s})] = 0$ (in the limit of infinite data), we get:

$$\begin{aligned} |\Delta_1(\boldsymbol{s})| \; &\leq \sum_{\boldsymbol{a}} \pi(\boldsymbol{a}|\boldsymbol{s}) |r_{\hat{M}}(\boldsymbol{s},\boldsymbol{a}) - r_M(\boldsymbol{s},\boldsymbol{a})| \leq \sum_{\boldsymbol{a}} \pi(\boldsymbol{a}|\boldsymbol{s}) \frac{C_{r,\delta}}{\sqrt{|\mathcal{D}(\boldsymbol{s})| \cdot |\mathcal{D}(\boldsymbol{a}|\boldsymbol{s})|}} \\ &= \frac{C_{r,\delta}}{\sqrt{|\mathcal{D}(\boldsymbol{s})|}} \sum_{\boldsymbol{a}} \frac{\pi(\boldsymbol{a}|\boldsymbol{s})}{\sqrt{\hat{\pi}_\beta(\boldsymbol{a}|\boldsymbol{s})}}, \end{aligned}$$

where the last step follows from the fact that $|\mathcal{D}(\boldsymbol{s},\boldsymbol{a})| = |\mathcal{D}(\boldsymbol{s})| \cdot \hat{\pi}_\beta(\boldsymbol{a}|\boldsymbol{s})$.

Next we bound the second term. We first note that if we can bound $||d^{\pi}_{\hat{M}} - d^{\pi}_{M}||_1$, (i.e., the total variation between the marginal state distributions in $\hat{M}$ and M, then we are done, since $|r_M(\boldsymbol{s}, \boldsymbol{a})| \leq R_{\max}$ and $\pi(\boldsymbol{a}|\boldsymbol{s}) \leq 1$, and hence we bound the second term effectively. We use an analysis similar to Achiam et al. [3] to obtain this total variation bound. Define, $G = (I - \gamma P^{\pi}_{M})^{-1}$ and $\boldsymbol{a}rG = (I - \gamma P^{\pi}_{\hat{M}})^{-1}$ and let $\Delta = P^{\pi}_{M} - P^{\pi}_{\hat{M}}$. Then, we can write:

$$d^{\pi}_{\hat{M}} - d^{\pi}_{M} = (1 - \gamma)(\boldsymbol{a}rG - G)\rho,$$

where $\rho(\boldsymbol{s})$ is the initial state distribution, which is assumed to be the same for both the MDPs. Then, using Equation 21 from Achiam et al. [3], we can simplify this to obtain,

$$d^{\pi}_{M} - d^{\pi}_{\hat{M}} = (1 - \gamma)\gamma G \Delta \boldsymbol{a}rG\mu = \gamma \boldsymbol{a}rG \Delta d^{\pi}_{\hat{M}}$$

Now, following steps similar to proof of Lemma 3 in Achiam et al. [3] we obtain,

$$\begin{aligned}
||\Delta d^{\pi}_{M}||_1 &= \sum_{\boldsymbol{s}'} \left| \sum_{\boldsymbol{s}} \Delta(\boldsymbol{s}'|\boldsymbol{s}) d^{\pi}_{M}(\boldsymbol{s}) \right| \leq \sum_{\boldsymbol{s},\boldsymbol{s}'} \left| \Delta(\boldsymbol{s}'|\boldsymbol{s}) \right| d^{\pi}_{\hat{M}} \\
&= \sum_{\boldsymbol{s},\boldsymbol{s}'} \left| \sum_{\boldsymbol{a}} \left(P_{\hat{M}}(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a}) - P_{M}(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a}) \right) \pi(\boldsymbol{a}|\boldsymbol{s}) \right| d^{\pi}_{\hat{M}}(\boldsymbol{s}) \\
&\leq \sum_{\boldsymbol{s},\boldsymbol{a}} ||P_{\hat{M}}(\cdot|\boldsymbol{s}, \boldsymbol{a}) - P_{M}(\cdot|\boldsymbol{s}, \boldsymbol{a})||_1 \pi(\boldsymbol{a}|\boldsymbol{s}) d^{\pi}_{\hat{M}}(\boldsymbol{s}) \\
&\leq \sum_{\boldsymbol{s}} d^{\pi}_{\hat{M}}(\boldsymbol{s}) \frac{C_{T,\delta}}{\sqrt{|\mathcal{D}(\boldsymbol{s})|}} \sum_{\boldsymbol{a}} \frac{\pi(\boldsymbol{a}|\boldsymbol{s})}{\sqrt{\hat{\pi}_{\beta}(\boldsymbol{a}|\boldsymbol{s})}}.
\end{aligned}$$

Hence, we can bound the second term by:

$$\left| \sum_{\boldsymbol{s}} \left(d^{\pi}_{\hat{M}}(\boldsymbol{s}) - d^{\pi}_{M}(\boldsymbol{s}) \right) \pi(\boldsymbol{a}|\boldsymbol{s}) r_M(\boldsymbol{s}, \boldsymbol{a}) \right| \leq \frac{\gamma C_{T,\delta} R_{\max}}{(1 - \gamma)} \sum_{\boldsymbol{s}} d^{\pi}_{\hat{M}}(\boldsymbol{s}) \frac{1}{\sqrt{|\mathcal{D}(\boldsymbol{s})|}} \sum_{\boldsymbol{a}} \frac{\pi(\boldsymbol{a}|\boldsymbol{s})}{\sqrt{\hat{\pi}_{\beta}(\boldsymbol{a}|\boldsymbol{s})}}.$$

To finally obtain $D_{\text{CQL}}(\pi, \pi_{\beta})(\boldsymbol{s})$ in the bound, let $\alpha(\boldsymbol{s}, \boldsymbol{a}) := \frac{\pi(\boldsymbol{a}|\boldsymbol{s})}{\sqrt{\hat{\pi}_{\beta}(\boldsymbol{a}|\boldsymbol{s})}}$. Then, we can write $D_{\text{CQL}}(\pi, \hat{\pi}_{\beta})(\boldsymbol{s})$ as follows:

$$\begin{aligned}
& D_{\text{CQL}}(\boldsymbol{s}) = \sum_{\boldsymbol{a}} \frac{\pi(\boldsymbol{a}|\boldsymbol{s})^2}{\hat{\pi}_{\beta}(\boldsymbol{a}|\boldsymbol{s})} - 1 \\
\implies & D_{\text{CQL}}(\boldsymbol{s}) + 1 = \sum_{\boldsymbol{a}} \alpha(\boldsymbol{s}, \boldsymbol{a})^2 \\
\implies & D_{\text{CQL}}(\boldsymbol{s}) + 1 \leq \left(\sum_{\boldsymbol{a}} \alpha(\boldsymbol{s}, \boldsymbol{a}) \right)^2 \leq |\mathcal{A}| \left(D_{\text{CQL}}(\boldsymbol{s}) + 1 \right).
\end{aligned}$$

Combining these arguments together, we obtain the following upper bound on $|J(\pi, M) - J(\pi, \hat{M})|$,

$$\left|J(\pi, \hat{M}) - J(\pi, M)\right| \leq \left(\frac{C_{r,\delta}}{1-\gamma} + \frac{\gamma R_{\max} C_{T,\delta}}{(1-\gamma)^2}\right) \mathbb{E}_{\boldsymbol{s} \sim d^{\pi}_{\hat{M}}(\boldsymbol{s})} \left[\frac{\sqrt{|\mathcal{A}|}}{\sqrt{|\mathcal{D}(\boldsymbol{s})|}} \sqrt{D_{\text{CQL}}(\pi, \hat{\pi}_\beta)(\boldsymbol{s}) + 1}\right].$$

□

The proof of Theorem C.4.6 is then completed by using the above Lemma for bounding the sampling error for π^* and then upper bounding the sampling error for $\hat{\pi}_\beta$ by the corresponding sampling error for π^*, hence giving us a factor of 2 on the sampling error term in Theorem C.4.6. To see why this is mathematically correct, note that $D_{\text{CQL}}(\hat{\pi}_\beta, \hat{\pi}_\beta)(\boldsymbol{s}) = 0$, hence $\sqrt{D_{\text{CQL}}(\pi^*, \hat{\pi}_\beta)(\boldsymbol{s}) + 1} \geq \sqrt{D_{\text{CQL}}(\hat{\pi}_\beta, \hat{\pi}_\beta)(\boldsymbol{s}) + 1}$, which means the sampling error term for π^* pointwise upper bounds the sampling error for $\hat{\pi}_\beta$, which justifies the factor of 2.

C.5 Extended Related Work

In this section, we discuss related works to supplement Section 4.6. Specifically, we discuss the relationships between CQL and uncertainty estimation and policy-constraint methods.

Relationship to uncertainty estimation in offline RL. A number of prior approaches to offline RL estimate some sort of epistemic uncertainty to determine the trustworthiness of a Q-value prediction [179, 95, 6, 204]. The policy is then optimized with respect to lower-confidence estimates derived using the uncertainty metric. However, it has been empirically noted that uncertainty-based methods are not sufficient to prevent against OOD actions [95, 179] in and of themselves, are often augmented with policy constraints due to the inability to estimate tight and calibrated uncertainty sets. Such loose or uncalibrated uncertainty sets are still effective in providing exploratory behavior in standard, online RL [254, 253], where these methods were originally developed. However, offline RL places high demands on the fidelity of such sets [204], making it hard to directly use these methods.

How does CQL relate to prior uncertainty estimation methods? Typical uncertainty estimation methods rely on learning a pointwise upper bound on the Q-function that depends on epistemic uncertainty [146, 253] and these upper-confidence bound values are then used for exploration. In the context of offline RL, this means learning a pointwise lower-bound on the Q-function. We show in Section 5.1.2 that, with a naïve choice of regularizer (Equation 5.1.1), we can learn a uniform lower-bound on the Q-function, however, we then showed that we can improve this bound since the value of the policy is the primary quantity of interest that needs to be lower-bounded. This

implies that CQL strengthens the popular practice of point-wise lower-bounds made by uncertainty estimation methods.

CAN WE MAKE CQL DEPENDENT ON UNCERTAINTY? We can slightly modify CQL to make it be account for epistemic uncertainty under certain statistical concentration assumptions. Typical uncertainty estimation methods in RL [254, 146] assume the applicability of concentration inequalities (for example, by making sub-Gaussian assumptions on the reward and dynamics), to obtain upper or lower-confidence bounds and the canonical amount of over- (under-) estimation is usually given by, $\mathcal{O}\left(\frac{1}{\sqrt{n(\boldsymbol{s},\boldsymbol{a})}}\right)$, where $n(\boldsymbol{s},\boldsymbol{a})$ is the number of times a state-action pair $(\boldsymbol{s},\boldsymbol{a})$ is observed in the dataset. We can incorporate such behavior in CQL by modifying Equation 5.1.3 to update Bellman error weighted by the cardinality of the dataset, $|\mathcal{D}|$, which gives rise to the following effective Q-function update in the tabular setting, without function approximation:

$$\hat{Q}^{k+1}(\boldsymbol{s},\boldsymbol{a}) = \mathcal{B}^{\pi}\hat{Q}^{k}(\boldsymbol{s},\boldsymbol{a}) - \alpha\frac{\mu(\boldsymbol{a}|\boldsymbol{s}) - \pi_{\beta}(\boldsymbol{a}|\boldsymbol{s})}{n(\boldsymbol{s},\boldsymbol{a})} \rightarrow \mathcal{B}^{\pi}\hat{Q}^{k}(\boldsymbol{s},\boldsymbol{a}) \;\text{ as }\; n(\boldsymbol{s},\boldsymbol{a}) \rightarrow \infty.$$

In the limit of infinite data, i.e. $n(\boldsymbol{s},\boldsymbol{a}) \rightarrow \infty$, we find that the amount of underestimation tends to 0. When only a finite-sized dataset is provided, i.e. $n(\boldsymbol{s},\boldsymbol{a}) < N$, for some N, we observe that by making certain assumptions, previously used in prior work [146, 254] on the concentration properties of the reward value, $r(\boldsymbol{s},\boldsymbol{a})$ and the dynamics function, $T(\boldsymbol{s}'|\boldsymbol{s},\boldsymbol{a})$, such as follows:

$$||\hat{r}(\boldsymbol{s},\boldsymbol{a}) - r(\boldsymbol{s},\boldsymbol{a})|| \leq \frac{C_r}{\sqrt{n(\boldsymbol{s},\boldsymbol{a})}} \quad \text{and} \quad ||\hat{T}(\boldsymbol{s}'|\boldsymbol{s},\boldsymbol{a}) - T(\boldsymbol{s}'|\boldsymbol{s},\boldsymbol{a})|| \leq \frac{C_T}{\sqrt{n(\boldsymbol{s},\boldsymbol{a})}},$$

where C_r and C_T are constants, that depend on the concentration properties of the MDP, and by appropriately choosing α, i.e. $\alpha = \Omega(n(\boldsymbol{s},\boldsymbol{a}))$, such that the learned Q-function still lower-bounds the actual Q-function (by nullifying the possible overestimation that appears due to finite samples), we are still guaranteed a lower bound.

C.6 ADDITIONAL EXPERIMENTAL SETUP AND IMPLEMENTATION DETAILS

In this section, we discuss some additional implementation details related to our method. As discussed in Section 5.1.5, CQL can be implemented as either a Q-learning or an actor-critic method. For our experiments on D4RL benchmarks [86], we implemented CQL on top of soft actor-critic (SAC) [122], and for experiments on discrete-action Atari tasks, we implemented CQL on top of QR-DQN [51]. We experimented with two ways of implementing CQL, first with a fixed α, where we chose $\alpha = 5.0$, and second with a varying α chosen via dual gradient-descent. The latter formulation automates the choice of α by introducing a "budget" parameter, τ, as shown below:

$$\min_{Q} \max_{\alpha \geq 0} \alpha \left(\mathbb{E}_{\boldsymbol{s}\sim d^{\pi_\beta}(\boldsymbol{s})} \left[\log \sum_{\boldsymbol{a}} \exp(Q(\boldsymbol{s},\boldsymbol{a})) - \mathbb{E}_{\boldsymbol{a}\sim\pi_\beta(\boldsymbol{a}|\boldsymbol{s})}\left[Q(\boldsymbol{s},\boldsymbol{a})\right] \right] - \tau \right) + \text{TD-Error}. \tag{C.6.1}$$

Equation C.6.1 implies that if the expected difference in Q-values is less than the specified threshold τ, α will adjust to be close to 0, whereas if the difference in Q-values is higher than the specified threshold, τ, then α is likely to take on high values, and thus more aggressively penalize Q-values. We refer to this version as CQL-Lagrange, and we found that this version drastically outperforms the fixed α version on the more complex AntMazes.

Choice of α. Our experiments in Section 4.5 use either a fixed penalty value $\alpha = 5.0$ or the Lagrange version to automatically tune α during training. For our experiments, across D4RL Gym MuJoCo domains, we choose a fixed value at $\alpha = 5.0$. For the other D4RL domains (Franka Kitchen and Adroit), we chose $\tau = 5.0$. And for our Atari experiments, we used a fixed penalty, with $\alpha = 1.0$ chosen uniformly for Table 3 (with 10% data), $\alpha = 4.0$ chosen uniformly for Table 3 (with 1% data), and $\alpha = 0.5$ for Figure 10.[1]

Computing $\log \sum_a \exp(Q(s,a))$. CQL($\mathcal{H}$) uses log-sum-exp in the objective for training the Q-function (Equation 5.1.4). In discrete action domains, we compute the log-sum-exp exactly by invoking the standard $\mathrm{tf.reduce\_logsumexp}()$ (or $\mathrm{torch.logsumexp}()$) functions provided by autodiff libraries. In continuous action tasks, CQL($\mathcal{H}$) uses importance sampling to compute this quantity, where in practice, we sampled **10** action samples at every state s from a uniform-at-random $\mathrm{Unif}(a)$ and **10** action samples from the current policy, $\pi(a|s)$ and used these alongside importance sampling to compute it as follows using $N = 10$ action samples:

$$\begin{aligned}
&\log \sum_a \exp(Q(s,a)) = \log \left(\frac{1}{2} \sum_a \exp(Q(s,a)) + \frac{1}{2} \sum_a \exp(Q(s,a)) \right) \\
&= \log \left(\frac{1}{2} \mathbb{E}_{a \sim \mathrm{Unif}(a)} \left[\frac{\exp(Q(s,a))}{\mathrm{Unif}(a)} \right] + \frac{1}{2} \mathbb{E}_{a \sim \pi(a|s)} \left[\frac{\exp(Q(s,a))}{\pi(a|s)} \right] \right) \\
&\approx \log \left(\frac{1}{2N} \sum_{a_i \sim \mathrm{Unif}(a)}^{N} \left[\frac{\exp(Q(s,a_i))}{\mathrm{Unif}(a)} \right] + \frac{1}{2N} \sum_{a_i \sim \pi(a|s)}^{N} \left[\frac{\exp(Q(s,a_i))}{\pi(a_i|s)} \right] \right).
\end{aligned}$$

On some domains, we observe that choosing higher values of N is better, since it is more effective in covering the space of Q-values and prevents overestimation. We can also choose to incorporate actions sampled from the policy at the next state into the computation of the log-sum-exp, however, this is not something that is necessarily needed.

Hyperparameters. For the updated results on the latest D4RL-v2 tasks, we used the CQL implementation at https://github.com/young-geng/CQL. We used a policy learning rate of 1e-4 and a critic learning rate of 3e-4 for the actor-critic version of CQL. This is to ensure

[1] For a standard DQN-style method that utilizes the mean-squared error backup, please the hyperparameters in: https://arxiv.org/abs/2112.04716. Notably, with a DQN-style architecture, $\alpha = 0.1$ is generally a good choice, across the board on all Atari games.

that the actor learning rate is smaller than the critic learning rate, which is required as per Theorem 5.1.3. Following the convention set by D4RL [86], we report the normalized, smooth average undiscounted return over 3 seeds for in our results in Section 4.5. The other hyperparameters we evaluated on during our preliminary experiments, and might be helpful guidelines for using CQL are as follows:

- **Lagrange threshold τ and α.** For the Gym-MuJoCo tasks, we used $\alpha = 5.0$, for the antmaze tasks, we used $\tau = 1.0$.
- **Number of gradient steps.** We evaluated our method on varying number of gradient steps. Since CQL generally uses a reduced policy learning rate (1e-4), we trained CQL methods for 1M gradient steps. For a number of the gym tasks (e.g., hopper and walker tasks), we find that CQL is trained in about 500k gradient steps, while for the halfcheetah tasks, they take 1M steps and are still improving. For instance on the halfcheetah-medium-expert dataset, we find that training for longer can give rise to better performance (e.g., $\approx 11k$ in return as compared to $\approx 9k$ as reported. Due to a lack of a proper valdiation error metric for offline Q-learning methods, deciding the number of gradient steps dynamically has been an open problem in offline RL [204]. Different prior methods choose the number of steps differently. For our experiments, we used 1M gradient steps for all D4RL domains, and followed the convention from Agarwal et al. [6], to report returns after 5x gradient steps of training for the Atari results in Table 3.
- **Choice of backup.** Instead of using an actor-critic version of CQL, we can instead also use an approximate max-backup for the Q-function in a continuous control setting. Similar to prior work [179], we sample 10 actions from the current policy at the next state $\boldsymbol{s}'$, called $\boldsymbol{a}_1, \cdots \boldsymbol{a}_{10} \sim \pi(\boldsymbol{a}'|\boldsymbol{s}')$ and generate the target values for the backup using the following equation: $r(\boldsymbol{s}, \boldsymbol{a}) + \gamma \max_{\boldsymbol{a}_1, \cdots, \boldsymbol{a}_{10}} Q(\boldsymbol{s}', \boldsymbol{a}')$. This is different from the standard actor-critic backup that performs an expectation of the Q-values $Q(\boldsymbol{s}', \boldsymbol{a}')$ at the next state under the policy's distribution $\pi(\boldsymbol{a}'|\boldsymbol{s}')$. We used this backup for the Antmaze domains.
- **Reward function.** For Antmaze tasks, we found it useful to not utilize a reward function of 0 and 1, but rather use shifted and scaled versions of these rewards. We used -5 and 5 as the two levels. This does not leak any form of domain knowledge into the rewards, but rather it enables the Q-function to propagate $+5$ faster into the initial states. On the other hand, a 0/1 reward is unable to propagate +1 into the Q-function at the initial state. To observe this, note that the change in the Q-function at the initial state-action pair due to a +1 at the end of a length 1000 trajectory is $\gamma^1 000 \times 1 = 4e - 5$, which is negligible. Understanding how the choice of the reward function affects optimization and error propagation in Q-learning is a subject of future work.

Other hyperparameters, were kept identical to SAC on the D4RL tasks, including the twin Q-function trick, soft-target updates, etc. We did observe that larger the size of the Q-network (3 or 4 hidden layers, compared to 2 hidden layers typically used by SAC), the better the performance. In the Atari domain, we based our implementation of CQL on top of the QR-DQN implementation provided by Agarwal et al. [6]. We did not tune any parameter from the QR-DQN implementation released with the official codebase with [6].

Model-Based Conservative Value Estimation

C.7 COMBO PROOFS FROM SECTION 5.2.4

In this section, we provide proofs for theoretical results in Section 5.2.4. Before the proofs, we note that all statements are proven in the case of finite state space (i.e., $|\mathcal{S}| < \infty$) and finite action space (i.e., $|\mathcal{A}| < \infty$) we define some commonly appearing notation symbols appearing in the proof:

- $P_{\mathcal{M}}$ and $r_{\mathcal{M}}$ (or P and r with no subscript for notational simplicity) denote the dynamics and reward function of the actual MDP $\mathcal{M}$
- $P_{\overline{\mathcal{M}}}$ and $r_{\overline{\mathcal{M}}}$ denote the dynamics and reward of the empirical MDP $\overline{\mathcal{M}}$ generated from the transitions in the dataset
- $P_{\widehat{\mathcal{M}}}$ and $r_{\widehat{\mathcal{M}}}$ denote the dynamics and reward of the MDP induced by the learned model $\widehat{\mathcal{M}}$

We also assume that whenever the cardinality of a particular state or state-action pair in the offline dataset $\mathcal{D}$, denoted by $|\mathcal{D}(\boldsymbol{s}, \boldsymbol{a})|$, appears in the denominator, we assume it is non-zero. For any non-existent $(\boldsymbol{s}, \boldsymbol{a}) \notin \mathcal{D}$, we can simply set $|\mathcal{D}(\boldsymbol{s}, \boldsymbol{a})|$ to be a small value < 1, which prevents any bound from producing trivially ∞ values.

C.7.1 *A Useful Lemma and Its Proof*

Before proving our main results, we first show that the penalty term in equation 5.2.3 is positive in expectation. Such a positive penalty is important to combat any overestimation that may arise as a result of using .

Lemma C.7.1 ((Interpolation Lemma). *For any $f \in [0, 1]$, and any given $\rho(\boldsymbol{s}, \boldsymbol{a}) \in \Delta^{|\mathcal{S}||\mathcal{A}|}$, let d_f be an f-interpolation of ρ and $\mathcal{D}$, i.e., $d_f(\boldsymbol{s}, \boldsymbol{a}) := fd(\boldsymbol{s}, \boldsymbol{a}) + (1 - f)\rho(\boldsymbol{s}, \boldsymbol{a})$. For a given iteration k of Equation 5.2.3, we restate the definition of the expected penalty under $\rho(\boldsymbol{s}, \boldsymbol{a})$ in Eq. 5.2.4:*

$$\nu(\rho, f) := \mathbb{E}_{\boldsymbol{s}, \boldsymbol{a} \sim \rho(\boldsymbol{s}, \boldsymbol{a})} \left[\frac{\rho(\boldsymbol{s}, \boldsymbol{a}) - d(\boldsymbol{s}, \boldsymbol{a})}{d_f(\boldsymbol{s}, \boldsymbol{a})} \right].$$

Then $\nu(\rho, f)$ satisfies, (1) $\nu(\rho, f) \geq 0$, $\forall \rho, f$, (2) $\nu(\rho, f)$ is monotonically increasing in f for a fixed ρ, and (3) $\nu(\rho, f) = 0$ iff $\forall\, \boldsymbol{s}, \boldsymbol{a},\ \rho(\boldsymbol{s}, \boldsymbol{a}) = d(\boldsymbol{s}, \boldsymbol{a})$ or $f = 0$.

Proof. To prove this lemma, we use algebraic manipulation on the expression for quantity $\nu(\rho, f)$ and show that it is indeed positive and monotonically increasing in $f \in [0, 1]$.

$$\begin{aligned}
\nu(\rho, f) &= \sum_{\boldsymbol{s},\boldsymbol{a}} \rho(\boldsymbol{s}, \boldsymbol{a}) \left(\frac{\rho(\boldsymbol{s}, \boldsymbol{a}) - d(\boldsymbol{s}, \boldsymbol{a})}{f d(\boldsymbol{s}, \boldsymbol{a}) + (1 - f)\rho(\boldsymbol{s}, \boldsymbol{a})} \right) \\
&= \sum_{\boldsymbol{s},\boldsymbol{a}} \rho(\boldsymbol{s}, \boldsymbol{a}) \left(\frac{\rho(\boldsymbol{s}, \boldsymbol{a}) - d(\boldsymbol{s}, \boldsymbol{a})}{\rho(\boldsymbol{s}, \boldsymbol{a}) + f(d(\boldsymbol{s}, \boldsymbol{a}) - \rho(\boldsymbol{s}, \boldsymbol{a}))} \right) \qquad \text{(C.7.1)} \\
\implies \frac{d\nu(\rho, f)}{df} &= \sum_{\boldsymbol{s},\boldsymbol{a}} \rho(\boldsymbol{s}, \boldsymbol{a}) \left(\rho(\boldsymbol{s}, \boldsymbol{a}) - d(\boldsymbol{s}, \boldsymbol{a})\right)^2 \cdot \left(\frac{1}{(\rho(\boldsymbol{s}, \boldsymbol{a}) + f(d(\boldsymbol{s}, \boldsymbol{a}) - \rho(\boldsymbol{s}, \boldsymbol{a}))} \right)^2 \geq 0 \\
&\qquad \forall f \in [0, 1]. \qquad \text{(C.7.2)}
\end{aligned}$$

Since the derivative of $\nu(\rho, f)$ with respect to f is always positive, it is an increasing function of f for a fixed ρ, and this proves the second part (2) of the Lemma. Using this property, we can show the part (1) of the Lemma as follows:

$$\begin{aligned}
\forall f \in (0, 1],\ \nu(\rho, f) \geq \nu(\rho, 0) = \sum_{\boldsymbol{s},\boldsymbol{a}} \rho(\boldsymbol{s}, \boldsymbol{a}) \frac{\rho(\boldsymbol{s}, \boldsymbol{a}) - d(\boldsymbol{s}, \boldsymbol{a})}{\rho(\boldsymbol{s}, \boldsymbol{a})} &= \sum_{\boldsymbol{s},\boldsymbol{a}} \left(\rho(\boldsymbol{s}, \boldsymbol{a}) - d(\boldsymbol{s}, \boldsymbol{a})\right) \\
&= 1 - 1 = 0. \qquad \text{(C.7.3)}
\end{aligned}$$

Finally, to prove the third part (3) of this Lemma, note that when $f = 0$, $\nu(\rho, f) = 0$ (as shown above), and similarly by setting $\rho(\boldsymbol{s}, \boldsymbol{a}) = d(\boldsymbol{s}, \boldsymbol{a})$ note that we obtain $\nu(\rho, f) = 0$. To prove the only if side of (3), assume that $f \neq 0$ and $\rho(\boldsymbol{s}, \boldsymbol{a}) \neq d(\boldsymbol{s}, \boldsymbol{a})$ and we will show that in this case $\nu(\rho, f) \neq 0$. When $d(\boldsymbol{s}, \boldsymbol{a}) \neq \rho(\boldsymbol{s}, \boldsymbol{a})$, the derivative $\frac{d\nu(\rho, f)}{df} > 0$ (i.e., strictly positive) and hence the function $\nu(\rho, f)$ is a strictly increasing function of f. Thus, in this case, $\nu(\rho, f) > 0 = \nu(\rho, 0)\ \forall f > 0$. Thus we have shown that if $\rho(\boldsymbol{s}, \boldsymbol{a}) \neq d(\boldsymbol{s}, \boldsymbol{a})$ and $f > 0$, $\nu(\rho, f) \neq 0$, which completes our proof for the only if side of (3). □

C.7.2 *Proof of Proposition 5.2.1*

Before proving this proposition, we provide a bound on the Bellman backup in the empirical MDP, $\mathcal{B}_{\overline{\mathcal{M}}}$. To do so, we formally define the standard concentration properties of the reward and transition dynamics in the empirical MDP, $\overline{\mathcal{M}}$, that we assume so as to prove Proposition C.7.1. Following prior work [253, 146, 181], we assume:

Assumption C.7.2. *$\forall\, \boldsymbol{s}, \boldsymbol{a} \in \mathcal{M}$, the following relationships hold with high probability, $\geq 1 - \delta$*

$$|r_{\overline{\mathcal{M}}}(\boldsymbol{s}, \boldsymbol{a}) - r(\boldsymbol{s}, \boldsymbol{a})| \leq \frac{C_{r,\delta}}{\sqrt{|\mathcal{D}(\boldsymbol{s}, \boldsymbol{a})|}}, \quad ||P_{\overline{\mathcal{M}}}(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a}) - P(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a})||_1 \leq \frac{C_{P,\delta}}{\sqrt{|\mathcal{D}(\boldsymbol{s}, \boldsymbol{a})|}}.$$

Under this assumption and assuming that the reward function in the MDP, $r(\boldsymbol{s}, \boldsymbol{a})$ is bounded, as $|r(\boldsymbol{s}, \boldsymbol{a})| \leq R_{\max}$, we can bound the difference between the empirical Bellman operator, $\mathcal{B}_{\overline{\mathcal{M}}}$ and the actual MDP, $\mathcal{B}_{\mathcal{M}}$,

$$\begin{aligned}
\left|\left(\mathcal{B}_{\overline{\mathcal{M}}}{}^{\pi}\hat{Q}^k\right) - \left(\mathcal{B}^{\pi}_{\mathcal{M}}\hat{Q}^k\right)\right| &= \left|(r_{\overline{\mathcal{M}}}(\boldsymbol{s}, \boldsymbol{a}) - r_{\mathcal{M}}(\boldsymbol{s}, \boldsymbol{a}))\right. \\
&\left. + \gamma \sum_{\boldsymbol{s}'} \left(P_{\overline{\mathcal{M}}}(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a}) - P_{\mathcal{M}}(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a})\right) \mathbb{E}_{\pi(\boldsymbol{a}'|\boldsymbol{s}')}\left[\hat{Q}^k(\boldsymbol{s}', \boldsymbol{a}')\right]\right| \\
&\leq \left|r_{\overline{\mathcal{M}}}(\boldsymbol{s}, \boldsymbol{a}) - r_{\mathcal{M}}(\boldsymbol{s}, \boldsymbol{a})\right| \\
&+ \gamma \left|\sum_{\boldsymbol{s}'} \left(P_{\overline{\mathcal{M}}}(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a}) - P_{\mathcal{M}}(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a})\right) \mathbb{E}_{\pi(\boldsymbol{a}'|\boldsymbol{s}')}\left[\hat{Q}^k(\boldsymbol{s}', \boldsymbol{a}')\right]\right| \\
&\leq \frac{C_{r,\delta} + \gamma C_{P,\delta} 2R_{\max}/(1-\gamma)}{\sqrt{|\mathcal{D}(\boldsymbol{s}, \boldsymbol{a})|}}.
\end{aligned}$$

Thus the overestimation due to sampling error in the empirical MDP, $\overline{\mathcal{M}}$ is bounded as a function of a bigger constant, $C_{r,P,\delta}$ that can be expressed as a function of $C_{r,\delta}$ and $C_{P,\delta}$, and depends on δ via a $\sqrt{\log(1/\delta)}$ dependency. For the purposes of proving Proposition C.7.3, we assume that:

$$\forall \boldsymbol{s}, \boldsymbol{a}, \quad \left|\left(\mathcal{B}_{\overline{\mathcal{M}}}{}^{\pi}\hat{Q}^k\right) - \left(\mathcal{B}^{\pi}_{\mathcal{M}}\hat{Q}^k\right)\right| \leq \frac{C_{r,T,\delta} R_{\max}}{(1-\gamma)\sqrt{|\mathcal{D}(\boldsymbol{s}, \boldsymbol{a})|}}. \tag{C.7.4}$$

Next, we provide a bound on the error between the bellman backup induced by the learned dynamics model and the learned reward, $\mathcal{B}_{\widehat{\mathcal{M}}}$, and the actual Bellman backup, $\mathcal{B}_{\mathcal{M}}$. To do so, we note that:

$$\begin{aligned}
\left|\left(\mathcal{B}_{\widehat{\mathcal{M}}}{}^{\pi}\hat{Q}^k\right) - \left(\mathcal{B}^{\pi}_{\mathcal{M}}\hat{Q}^k\right)\right| &= \left|(r_{\widehat{\mathcal{M}}}(\boldsymbol{s}, \boldsymbol{a}) - r_{\mathcal{M}}(\boldsymbol{s}, \boldsymbol{a}))\right. \qquad \text{(C.7.5)} \\
&\left. + \gamma \sum_{\boldsymbol{s}'} \left(P_{\widehat{\mathcal{M}}}(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a}) - P_{\mathcal{M}}(\boldsymbol{s}'|\boldsymbol{s}, \boldsymbol{a})\right) \mathbb{E}_{\pi(\boldsymbol{a}'|\boldsymbol{s}')}\left[\hat{Q}^k(\boldsymbol{s}', \boldsymbol{a}')\right]\right| \\
&\leq \left|r_{\widehat{\mathcal{M}}}(\boldsymbol{s}, \boldsymbol{a}) - r_{\mathcal{M}}(\boldsymbol{s}, \boldsymbol{a})\right| + \gamma \frac{2R_{\max}}{1-\gamma} D(P, P_{\widehat{\mathcal{M}}}), \qquad \text{(C.7.6)}
\end{aligned}$$

where $D(P, P_{\widehat{\mathcal{M}}})$ is the total-variation divergence between the learned dynamics model and the actual MDP. Now, we show that the asymptotic Q-function learned by COMBO lower-bounds the actual Q-function of any policy π with high probability for a large enough $\beta \geq 0$. We will use Equations E.2.3 and C.7.6 to prove such a result.

Proposition C.7.3 (Asymptotic lower-bound)**.** *Let P^{π} denote the Hadamard product of the dynamics P and a given policy π in the actual MDP and let $S^{\pi} := (I - \gamma P^{\pi})^{-1}$. Let D denote the total-variation divergence between two probability distributions. For any $\pi(\boldsymbol{a}|\boldsymbol{s})$, the Q-function*

obtained by recursively applying Equation 5.2.3, with $\hat{\mathcal{B}}^\pi = f\mathcal{B}^\pi_{\mathcal{M}} + (1-f)\mathcal{B}^\pi_{\widehat{\mathcal{M}}}$*, with probability at least* $1-\delta$*, results in* $\hat{Q}^\pi$ *that satisfies:*

$$\forall \boldsymbol{s}, \boldsymbol{a},\ \hat{Q}^\pi(\boldsymbol{s},\boldsymbol{a}) \leq Q^\pi(\boldsymbol{s},\boldsymbol{a}) - \beta \cdot \left[S^\pi\left[\frac{\rho - d}{d_f}\right]\right](\boldsymbol{s},\boldsymbol{a}) + f\left[S^\pi\left[\frac{C_{r,T,\delta}R_{\max}}{(1-\gamma)\sqrt{|\mathcal{D}|}}\right]\right](\boldsymbol{s},\boldsymbol{a}) + (1-f)\left[S^\pi\left[|r - r_{\widehat{\mathcal{M}}}| + \frac{2\gamma R_{\max}}{1-\gamma} D(P, P_{\widehat{\mathcal{M}}})\right]\right](\boldsymbol{s},\boldsymbol{a}).$$

Proof. We first note that the Bellman backup $\hat{\mathcal{B}}^\pi$ induces the following Q-function iterates as per Equation 5.2.3,

$$\begin{aligned}
\hat{Q}^{k+1}(\boldsymbol{s},\boldsymbol{a}) &= \left(\hat{\mathcal{B}}^\pi \hat{Q}^k\right)(\boldsymbol{s},\boldsymbol{a}) - \beta\frac{\rho(\boldsymbol{s},\boldsymbol{a}) - d(\boldsymbol{s},\boldsymbol{a})}{d_f(\boldsymbol{s},\boldsymbol{a})} \\
&= f\left(\mathcal{B}^\pi_{\mathcal{M}}\hat{Q}^k\right)(\boldsymbol{s},\boldsymbol{a}) + (1-f)\left(\mathcal{B}^\pi_{\widehat{\mathcal{M}}}\hat{Q}^k\right)(\boldsymbol{s},\boldsymbol{a}) - \beta\frac{\rho(\boldsymbol{s},\boldsymbol{a}) - d(\boldsymbol{s},\boldsymbol{a})}{d_f(\boldsymbol{s},\boldsymbol{a})} \\
&= \left(\mathcal{B}^\pi \hat{Q}^k\right)(\boldsymbol{s},\boldsymbol{a}) - \beta\frac{\rho(\boldsymbol{s},\boldsymbol{a}) - d(\boldsymbol{s},\boldsymbol{a})}{d_f(\boldsymbol{s},\boldsymbol{a})} + (1-f)\left(\mathcal{B}_{\widehat{\mathcal{M}}}{}^\pi \hat{Q}^k - \mathcal{B}^\pi \hat{Q}^k\right)(\boldsymbol{s},\boldsymbol{a}) \\
&\quad + f\left(\mathcal{B}_{\mathcal{M}}{}^\pi \hat{Q}^k - \mathcal{B}^\pi \hat{Q}^k\right)(\boldsymbol{s},\boldsymbol{a}) \\
\hat{Q}^{k+1} &\leq \left(\mathcal{B}^\pi \hat{Q}^k\right) - \beta\frac{\rho - d}{d_f} + (1-f)\left[|r_{\widehat{\mathcal{M}}} - r_{\mathcal{M}}| + \frac{2\gamma R_{\max} D(P, P_{\widehat{\mathcal{M}}})}{1-\gamma}\right] + \frac{f C_{r,T,\delta} R_{\max}}{(1-\gamma)\sqrt{|\mathcal{D}|}}
\end{aligned}$$

Since the RHS upper bounds the Q-function pointwise for each $(\boldsymbol{s},\boldsymbol{a})$, the fixed point of the Bellman iteration process will be pointwise smaller than the fixed point of the Q-function found by solving for the RHS via equality. Thus, we get that

$$\hat{Q}^\pi(\boldsymbol{s},\boldsymbol{a}) \leq \underbrace{S^\pi r_{\mathcal{M}}}_{=Q^\pi(\boldsymbol{s},\boldsymbol{a})} - \beta\left[S^\pi\left[\frac{\rho - d}{d_f}\right]\right](\boldsymbol{s},\boldsymbol{a}) + f\left[S^\pi\left[\frac{C_{r,T,\delta}R_{\max}}{(1-\gamma)\sqrt{|\mathcal{D}|}}\right]\right](\boldsymbol{s},\boldsymbol{a}) + (1-f)\left[S^\pi\left[|r - r_{\widehat{\mathcal{M}}}| + \frac{2\gamma R_{\max}}{1-\gamma} D(P, P_{\widehat{\mathcal{M}}})\right]\right](\boldsymbol{s},\boldsymbol{a}),$$

which completes the proof of this proposition. □

Next, we use the result and proof technique from Proposition C.7.3 to prove Corollary 5.2.1, that in expectation under the initial state-distribution, the expected Q-value is indeed a lower-bound.

Corollary C.7.4 (Corollary 5.2.1 restated)**.** *For a sufficiently large* β*, we have a lower-bound that* $\mathbb{E}_{\boldsymbol{s}\sim\mu_0,\boldsymbol{a}\sim\pi(\cdot|\boldsymbol{s})}[\hat{Q}^\pi(\boldsymbol{s},\boldsymbol{a})] \leq \mathbb{E}_{\boldsymbol{s}\sim\mu_0,\boldsymbol{a}\sim\pi(\cdot|\boldsymbol{s})}[Q^\pi(\boldsymbol{s},\boldsymbol{a})]$*, where* $\mu_0(\boldsymbol{s})$ *is the initial state distribution. Furthermore, when* ϵ_s *is small, such as in the large sample regime; or when the model bias* ϵ_m *is small, a small* β *is sufficient along with an appropriate choice of* f*.*

Proof. To prove this corollary, we note a slightly different variant of Proposition C.7.3. To observe this, we will deviate from the proof of Proposition C.7.3 slightly and will aim to express the inequality using $\mathcal{B}_{\widehat{\mathcal{M}}}$, the Bellman operator defined by the learned model and the reward function. Denoting $(I - \gamma P_{\widehat{\mathcal{M}}})^{-1}$ as $S^{\pi}_{\widehat{\mathcal{M}}}$, doing this will intuitively allow us to obtain $\beta \left(\mu(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s})\right)^T \left(S^{\pi}_{\widehat{\mathcal{M}}}\left[\frac{\rho - d}{d_f}\right]\right)(\boldsymbol{s}, \boldsymbol{a})$ as the conservative penalty which can be controlled by choosing β appropriately so as to nullify the potential overestimation caused due to other terms. Formally,

$$\hat{Q}^{k+1}(\boldsymbol{s}, \boldsymbol{a}) = \left(\hat{\mathcal{B}}^{\pi}\hat{Q}^k\right)(\boldsymbol{s}, \boldsymbol{a}) - \beta\frac{\rho(\boldsymbol{s}, \boldsymbol{a}) - d(\boldsymbol{s}, \boldsymbol{a})}{d_f(\boldsymbol{s}, \boldsymbol{a})} = \left(\mathcal{B}^{\pi}_{\widehat{\mathcal{M}}}\hat{Q}^k\right)(\boldsymbol{s}, \boldsymbol{a}) - \beta\frac{\rho(\boldsymbol{s}, \boldsymbol{a}) - d(\boldsymbol{s}, \boldsymbol{a})}{d_f(\boldsymbol{s}, \boldsymbol{a})} + f\underbrace{\left(\mathcal{B}^{\pi}_{\mathcal{M}} - \mathcal{B}^{\pi}_{\widehat{\mathcal{M}}}\hat{Q}^k\right)(\boldsymbol{s}, \boldsymbol{a})}_{:=\Delta(\boldsymbol{s}, \boldsymbol{a})}$$

By controlling $\Delta(\boldsymbol{s}, \boldsymbol{a})$ using the pointwise triangle inequality:

$$\forall \boldsymbol{s}, \boldsymbol{a}, \quad \left|\mathcal{B}^{\pi}_{\mathcal{M}}\hat{Q}^k - \mathcal{B}^{\pi}_{\widehat{\mathcal{M}}}\hat{Q}^k\right| \leq \left|\mathcal{B}^{\pi}\hat{Q}^k - \mathcal{B}^{\pi}_{\widehat{\mathcal{M}}}\hat{Q}^k\right| + \left|\mathcal{B}^{\pi}_{\mathcal{M}}\hat{Q}^k - \mathcal{B}^{\pi}\hat{Q}^k\right|, \tag{C.7.7}$$

and then iterating the backup $\mathcal{B}^{\pi}_{\widehat{\mathcal{M}}}$ to its fixed point and finally noting that $\rho(\boldsymbol{s}, \boldsymbol{a}) = \left((\mu \cdot \pi)^T S^{\pi}_{\widehat{\mathcal{M}}}\right)(\boldsymbol{s}, \boldsymbol{a})$, we obtain:

$$\mathbb{E}_{\mu,\pi}[\hat{Q}^{\pi}(\boldsymbol{s}, \boldsymbol{a})] \leq \mathbb{E}_{\mu,\pi}[Q^{\pi}_{\widehat{\mathcal{M}}}(\boldsymbol{s}, \boldsymbol{a})] - \beta\,\mathbb{E}_{\rho(\boldsymbol{s},\boldsymbol{a})}\left[\frac{\rho(\boldsymbol{s}, \boldsymbol{a}) - d(\boldsymbol{s}, \boldsymbol{a})}{d_f(\boldsymbol{s}, \boldsymbol{a})}\right] + \text{terms independent of } \beta. \tag{C.7.8}$$

The terms marked as "terms independent of β" correspond to the additional positive error terms obtained by iterating $\left|\mathcal{B}^{\pi}\hat{Q}^k - \mathcal{B}^{\pi}_{\widehat{\mathcal{M}}}\hat{Q}^k\right|$ and $\left|\mathcal{B}^{\pi}_{\mathcal{M}}\hat{Q}^k - \mathcal{B}^{\pi}\hat{Q}^k\right|$, which can be bounded similar to the proof of Proposition C.7.3 above. Now by replacing the model Q-function, $\mathbb{E}_{\mu,\pi}[Q^{\pi}_{\widehat{\mathcal{M}}}(\boldsymbol{s}, \boldsymbol{a})]$ with the actual Q-function, $\mathbb{E}_{\mu,\pi}[Q^{\pi}(\boldsymbol{s}, \boldsymbol{a})]$ and adding an error term corresponding to model error to the bound, we obtain that:

$$\mathbb{E}_{\mu,\pi}[\hat{Q}^{\pi}(\boldsymbol{s}, \boldsymbol{a})] \leq \mathbb{E}_{\mu,\pi}[Q^{\pi}(\boldsymbol{s}, \boldsymbol{a})] + \text{terms independent of } \beta - \beta\underbrace{\mathbb{E}_{\rho(\boldsymbol{s},\boldsymbol{a})}\left[\frac{\rho(\boldsymbol{s}, \boldsymbol{a}) - d(\boldsymbol{s}, \boldsymbol{a})}{d_f(\boldsymbol{s}, \boldsymbol{a})}\right]}_{=\nu(\rho,f)>0}. \tag{C.7.9}$$

Hence, by choosing β large enough, we obtain the desired lower bound guarantee. □

Remark C.7.5 (COMBO does not underestimate at every $\boldsymbol{s} \in \mathcal{D}$ unlike CQL.). *Before concluding this section, we discuss how the bound obtained by COMBO (Equation C.7.9) is tighter than CQL. CQL learns a Q-function such that the value of the policy under the resulting Q-function lower-bounds the true value function at each state $\boldsymbol{s} \in \mathcal{D}$ individually (in*

the absence of no sampling error), i.e., $\forall \boldsymbol{s} \in \mathcal{D}, \hat{V}^{\pi}_{CQL}(\boldsymbol{s}) \leq V^{\pi}(\boldsymbol{s})$, whereas the bound in COMBO is only valid in expectation of the value function over the initial state distribution, i.e., $\mathbb{E}_{\boldsymbol{s}\sim\mu_0(\boldsymbol{s})}[\hat{V}^{\pi}_{COMBO}(\boldsymbol{s})] \leq \mathbb{E}_{\boldsymbol{s}\sim\mu_0(\boldsymbol{s})}[V^{\pi}(\boldsymbol{s})]$, and the value function at a given state may not be a lower-bound. For instance, COMBO can overestimate the value of a state more frequent in the dataset distribution $d(\boldsymbol{s}, \boldsymbol{a})$ but not so frequent in the $\rho(\boldsymbol{s}, \boldsymbol{a})$ marginal distribution of the policy under the learned model $\widehat{\mathcal{M}}$. To see this more formally, note that the expected penalty added in the effective Bellman backup performed by COMBO (Equation 5.2.3), in expectation under the dataset distribution $d(\boldsymbol{s}, \boldsymbol{a})$, $\widetilde{\nu}(\rho, d, f)$ is actually ***negative****:*

$$\widetilde{\nu}(\rho, d, f) = \sum_{\boldsymbol{s},\boldsymbol{a}} d(\boldsymbol{s}, \boldsymbol{a}) \frac{\rho(\boldsymbol{s}, \boldsymbol{a}) - d(\boldsymbol{s}, \boldsymbol{a})}{d_f(\boldsymbol{s}, \boldsymbol{a})} = -\sum_{\boldsymbol{s},\boldsymbol{a}} d(\boldsymbol{s}, \boldsymbol{a}) \frac{d(\boldsymbol{s}, \boldsymbol{a}) - \rho(\boldsymbol{s}, \boldsymbol{a})}{f d(\boldsymbol{s}, \boldsymbol{a}) + (1-f)\rho(\boldsymbol{s}, \boldsymbol{a})} < 0,$$

where the final inequality follows via a direct application of the proof of Lemma C.7.1. Thus, COMBO actually overestimates *the values at atleast some states (in the dataset) unlike CQL.*

C.7.3 *Proof of Proposition 5.2.2*

In this section, we will provide a proof for Proposition 5.2.2, and show that the COMBO can be less conservative in terms of the estimated value. To recall, let $\Delta^{\pi}_{\text{COMBO}} := \mathbb{E}_{\boldsymbol{s},\boldsymbol{a}\sim d_{\overline{\mathcal{M}}}(\boldsymbol{s}),\pi(\boldsymbol{a}|\boldsymbol{s})}\left[\hat{Q}^{\pi}(\boldsymbol{s}, \boldsymbol{a}\right]$ and let $\Delta^{\pi}_{\text{CQL}} := \mathbb{E}_{\boldsymbol{s},\boldsymbol{a}\sim d_{\overline{\mathcal{M}}},\pi(\boldsymbol{a}|\boldsymbol{s})}\left[\hat{Q}^{\pi}_{\text{CQL}}(\boldsymbol{s}, \boldsymbol{a})\right]$. From Kumar et al. [181], we obtain that $\hat{Q}^{\pi}_{\text{CQL}}(\boldsymbol{s}, \boldsymbol{a}) := Q^{\pi}(\boldsymbol{s}, \boldsymbol{a}) - \beta\frac{\pi(\boldsymbol{a}|\boldsymbol{s}) - \pi_\beta(\boldsymbol{a}|\boldsymbol{s})}{\pi_\beta(\boldsymbol{a}|\boldsymbol{s})}$. We shall derive the condition for the real data fraction $f = 1$ for COMBO, thus making sure that $d_f(\boldsymbol{s}) = d^{\pi_\beta}(\boldsymbol{s})$. To derive the condition when $\Delta^{\pi}_{\text{COMBO}} \geq \Delta^{\pi}_{\text{CQL}}$, we note the following simplifications:

$$\Delta^{\pi}_{\text{COMBO}} \geq \Delta^{\pi}_{\text{CQL}} \tag{C.7.10}$$

$$\implies \sum_{\boldsymbol{s},\boldsymbol{a}} d_{\overline{\mathcal{M}}}(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s})\hat{Q}^{\pi}(\boldsymbol{s}, \boldsymbol{a}) \geq \sum_{\boldsymbol{s},\boldsymbol{a}} d_{\overline{\mathcal{M}}}(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s})\hat{Q}^{\pi}_{\text{CQL}}(\boldsymbol{s}, \boldsymbol{a}) \tag{C.7.11}$$

$$\implies \beta \sum_{\boldsymbol{s},\boldsymbol{a}} d_{\overline{\mathcal{M}}}(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s}) \left(\frac{\rho(\boldsymbol{s}, \boldsymbol{a}) - d^{\pi_\beta}(\boldsymbol{s})\pi_\beta(\boldsymbol{a}|\boldsymbol{s})}{d^{\pi_\beta}(\boldsymbol{s})\pi_\beta(\boldsymbol{a}|\boldsymbol{s})}\right) \tag{C.7.12}$$

$$\leq \beta \sum_{\boldsymbol{s},\boldsymbol{a}} d_{\overline{\mathcal{M}}}(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s}) \left(\frac{\pi(\boldsymbol{a}|\boldsymbol{s}) - \pi_\beta(\boldsymbol{a}|\boldsymbol{s})}{\pi_\beta(\boldsymbol{a}|\boldsymbol{s})}\right). \tag{C.7.13}$$

Now, in the expression on the left-hand side, we add and subtract $d^{\pi_\beta}(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s})$ from the numerator inside the paranthesis.

$$\sum_{\boldsymbol{s},\boldsymbol{a}} d_{\overline{\mathcal{M}}}(\boldsymbol{s},\boldsymbol{a}) \left(\frac{\rho(\boldsymbol{s},\boldsymbol{a}) - d^{\pi_\beta}(\boldsymbol{s})\pi_\beta(\boldsymbol{a}|\boldsymbol{s})}{d^{\pi_\beta}(\boldsymbol{s})\pi_\beta(\boldsymbol{a}|\boldsymbol{s})} \right) \tag{C.7.14}$$

$$= \sum_{\boldsymbol{s},\boldsymbol{a}} d_{\overline{\mathcal{M}}}(\boldsymbol{s},\boldsymbol{a}) \left(\frac{\rho(\boldsymbol{s},\boldsymbol{a}) - d^{\pi_\beta}(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s}) + d^{\pi_\beta}(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s}) - d^{\pi_\beta}(\boldsymbol{s})\pi_\beta(\boldsymbol{a}|\boldsymbol{s})}{d^{\pi_\beta}(\boldsymbol{s})\pi_\beta(\boldsymbol{a}|\boldsymbol{s})} \right) \tag{C.7.15}$$

$$= \underbrace{\sum_{\boldsymbol{s},\boldsymbol{a}} d_{\overline{\mathcal{M}}}(\boldsymbol{s},\boldsymbol{a}) \frac{\pi(\boldsymbol{a}|\boldsymbol{s}) - \pi_\beta(\boldsymbol{a}|\boldsymbol{s})}{\pi_\beta(\boldsymbol{a}|\boldsymbol{s})}}_{(1)} + \sum_{\boldsymbol{s},\boldsymbol{a}} d_{\overline{\mathcal{M}}}(\boldsymbol{s},\boldsymbol{a}) \cdot \frac{\rho(\boldsymbol{s}) - d^{\pi_\beta}(\boldsymbol{s})}{d^{\pi_\beta}(\boldsymbol{s})} \cdot \frac{\pi(\boldsymbol{a}|\boldsymbol{s})}{\pi_\beta(\boldsymbol{a}|\boldsymbol{s})} \tag{C.7.16}$$

The term marked (1) is identical to the CQL term that appears on the right in Equation C.7.12. Thus the inequality in Equation C.7.12 is satisfied when the second term above is negative. To show this, first note that $d^{\pi_\beta}(\boldsymbol{s}) = d_{\overline{\mathcal{M}}}(\boldsymbol{s})$ which results in a cancellation. Finally, re-arranging the second term into expectations gives us the desired result. An analogous condition can be derived when $f \neq 1$, but we omit that derivation as it will be hard to interpret terms appear in the final inequality.

C.7.4 *Proof of Proposition 5.2.3*

To prove the policy improvement result in Proposition 5.2.3, we first observe that using Equation 5.2.3 for Bellman backups amounts to finding a policy that maximizes the return of the policy in the a modified "f-interpolant" MDP which admits the Bellman backup $^\pi$, and is induced by a linear interpolation of backups in the empirical MDP $\overline{\mathcal{M}}$ and the MDP induced by a dynamics model $\widehat{\mathcal{M}}$ and the return of a policy π in this effective f-interpolant MDP is denoted by $J(\overline{\mathcal{M}}, \widehat{\mathcal{M}}, f, \pi)$. Alongside this, the return is penalized by the conservative penalty where ρ^π denotes the marginal state-action distribution of policy π in the learned model $\widehat{\mathcal{M}}$.

$$\hat{J}(f,\pi) = J(\overline{\mathcal{M}}, \widehat{\mathcal{M}}, f, \pi) - \beta \frac{\nu(\rho^\pi, f)}{1-\gamma}. \tag{C.7.17}$$

We will require bounds on the return of a policy π in this f-interpolant MDP, $J(\overline{\mathcal{M}}, \widehat{\mathcal{M}}, f, \pi)$, which we first prove separately as Lemma C.7.6 below and then move to the proof of Proposition 5.2.3.

Lemma C.7.6 (Bound on return in f-interpolant MDP)**.** *For any two MDPs, $\mathcal{M}_1$ and $\mathcal{M}_2$, with the same state-space, action-space and discount factor, and for a given fraction $f \in [0,1]$, define the f-interpolant MDP $\mathcal{M}_f$ as the MDP on the same state-space, action-space and with the same discount as the MDP with dynamics: $P_{\mathcal{M}_f} := fP_{\mathcal{M}_1} + (1-f)P_{\mathcal{M}_2}$ and reward function:*

$r_{\mathcal{M}_f} := f r_{\mathcal{M}_1} + (1-f) r_{\mathcal{M}_2}$. *Then, given any auxiliary MDP, $\mathcal{M}$, the return of any policy π in $\mathcal{M}_f$, $J(\pi, \mathcal{M}_f)$, also denoted by $J(\mathcal{M}_1, \mathcal{M}_2, f, \pi)$, lies in the interval:*

$$\left[J(\pi, \mathcal{M}) - \alpha, \;\; J(\pi, \mathcal{M}) + \alpha\right], \qquad \textit{where } \alpha \textit{ is given by:}$$

$$\begin{aligned}\alpha = \frac{2\gamma(1-f)}{(1-\gamma)^2} R_{\max} D\left(P_{\mathcal{M}_2}, P_{\mathcal{M}}\right) + \frac{\gamma f}{1-\gamma} \left| \mathbb{E}_{d^\pi_{\mathcal{M}} \pi} \left[\left(P^\pi_{\mathcal{M}} - P^\pi_{\mathcal{M}_1} \right) Q^\pi_{\mathcal{M}} \right] \right| \\ + \frac{f}{1-\gamma} \mathbb{E}_{\boldsymbol{s},\boldsymbol{a} \sim d^\pi_{\mathcal{M}} \pi} [|r_{\mathcal{M}_1}(\boldsymbol{s},\boldsymbol{a}) - r_{\mathcal{M}}(\boldsymbol{s},\boldsymbol{a})|] + \frac{1-f}{1-\gamma} \mathbb{E}_{\boldsymbol{s},\boldsymbol{a} \sim d^\pi_{\mathcal{M}} \pi} [|r_{\mathcal{M}_2}(\boldsymbol{s},\boldsymbol{a}) - r_{\mathcal{M}}(\boldsymbol{s},\boldsymbol{a})|].\end{aligned} \tag{C.7.18}$$

Proof. To prove this lemma, we note two general inequalities. First, note that for a fixed transition dynamics, say P, the return decomposes linearly in the components of the reward as the expected return is linear in the reward function:

$$J(P, r_{\mathcal{M}_f}) = J(P, f r_{\mathcal{M}_1} + (1-f) r_{\mathcal{M}_2}) = f J(P, r_{\mathcal{M}_1}) + (1-f) J(P, r_{\mathcal{M}_2}).$$

As a result, we can bound $J(P, r_{\mathcal{M}_f})$ using $J(P, r)$ for a new reward function r of the auxiliary MDP, $\mathcal{M}$, as follows

$$\begin{aligned} J(P, r_{\mathcal{M}_f}) &= J(P, f r_{\mathcal{M}_1} + (1-f) r_{\mathcal{M}_2}) = J(P, r + f(r_{\mathcal{M}_1} - r) + (1-f)(r_{\mathcal{M}_2} - r) \\ &= J(P, r) + f J(P, r_{\mathcal{M}_1} - r) + (1-f) J(P, r_{\mathcal{M}_2} - r) \\ &= J(P, r) + \frac{f}{1-\gamma} \mathbb{E}_{\boldsymbol{s},\boldsymbol{a} \sim d^\pi_{\mathcal{M}}(\boldsymbol{s}) \pi(\boldsymbol{a}|\boldsymbol{s})} \left[r_{\mathcal{M}_1}(\boldsymbol{s},\boldsymbol{a}) - r(\boldsymbol{s},\boldsymbol{a}) \right] \\ &\quad + \frac{1-f}{1-\gamma} \mathbb{E}_{\boldsymbol{s},\boldsymbol{a} \sim d^\pi_{\mathcal{M}}(\boldsymbol{s}) \pi(\boldsymbol{a}|\boldsymbol{s})} \left[r_{\mathcal{M}_2}(\boldsymbol{s},\boldsymbol{a}) - r(\boldsymbol{s},\boldsymbol{a}) \right]. \end{aligned}$$

Second, note that for a given reward function, r, but a linear combination of dynamics, the following bound holds:

$$\begin{aligned} J(P_{\mathcal{M}_f}, r) &= J(f P_{\mathcal{M}_1} + (1-f) P_{\mathcal{M}_2}, r) \\ &= J(P_{\mathcal{M}} + f(P_{\mathcal{M}_1} - P_{\mathcal{M}}) + (1-f)(P_{\mathcal{M}_2} - P_{\mathcal{M}}), r) \\ &= J(P_{\mathcal{M}}, r) - \frac{\gamma(1-f)}{1-\gamma} \mathbb{E}_{\boldsymbol{s},\boldsymbol{a} \sim d^\pi_{\mathcal{M}}(\boldsymbol{s}) \pi(\boldsymbol{a}|\boldsymbol{s})} \left[\left(P^\pi_{\mathcal{M}_2} - P^\pi_{\mathcal{M}} \right) Q^\pi_{\mathcal{M}} \right] \\ &\quad - \frac{\gamma f}{1-\gamma} \mathbb{E}_{\boldsymbol{s},\boldsymbol{a} \sim d^\pi_{\mathcal{M}}(\boldsymbol{s}) \pi(\boldsymbol{a}|\boldsymbol{s})} \left[\left(P^\pi_{\mathcal{M}} - P^\pi_{\mathcal{M}_1} \right) Q^\pi_{\mathcal{M}} \right] \\ &\in \left[J(P_{\mathcal{M}}, r) \;\pm\; \left(\frac{\gamma f}{(1-\gamma)} \left| \mathbb{E}_{\boldsymbol{s},\boldsymbol{a} \sim d^\pi_{\mathcal{M}}(\boldsymbol{s}) \pi(\boldsymbol{a}|\boldsymbol{s})} \left[\left(P^\pi_{\mathcal{M}} - P^\pi_{\mathcal{M}_1} \right) Q^\pi_{\mathcal{M}} \right] \right| \right. \right. \\ &\quad \left. \left. + \frac{2\gamma(1-f) R_{\max}}{(1-\gamma)^2} D(P_{\mathcal{M}_2}, P_{\mathcal{M}}) \right) \right]. \end{aligned}$$

To observe the third equality, we utilize the result on the difference between returns of a policy π on two different MDPs, $P_{\mathcal{M}_1}$ and $P_{\mathcal{M}_f}$ from Agarwal et al. [5] (Chapter 2, Lemma 2.2, Simulation Lemma), and additionally incorporate the auxiliary MDP $\mathcal{M}$ in the expression via addition and subtraction in the previous (second) step. In the fourth step, we finally bound one term that corresponds to the learned model via the total-variation divergence $D(P_{\mathcal{M}_2}, P_{\mathcal{M}})$ and the other term corresponding to the empirical MDP $\overline{\mathcal{M}}$ is left in its expectation form to be bounded later.

Using the above bounds on return for reward-mixtures and dynamics-mixtures, proving this lemma is straightforward:

$$J(\mathcal{M}_1, \mathcal{M}_2, f, \pi) := J(P_{\mathcal{M}_f}, fr_{\mathcal{M}_1} + (1-f)r_{\mathcal{M}_2}) = J(fP_{\mathcal{M}_1} + (1-f)P_{\mathcal{M}_2}, r_{\mathcal{M}_f})$$
$$\in \Big[J(P_{\mathcal{M}_f}, r_{\mathcal{M}}) \pm \underbrace{\left(\frac{f}{1-\gamma} \mathbb{E}_{\boldsymbol{s},\boldsymbol{a} \sim d^{\pi}_{\mathcal{M}} \pi}[|r_{\mathcal{M}_1}(\boldsymbol{s},\boldsymbol{a}) - r_{\mathcal{M}}(\boldsymbol{s},\boldsymbol{a})|] + \frac{1-f}{1-\gamma} \mathbb{E}_{\boldsymbol{s},\boldsymbol{a} \sim d^{\pi}_{\mathcal{M}} \pi}[|r_{\mathcal{M}_2}(\boldsymbol{s},\boldsymbol{a}) - r_{\mathcal{M}}(\boldsymbol{s},\boldsymbol{a})|] \right)}_{:=\Delta_R} \Big],$$

where the second step holds via linear decomposition of the return of π in $\mathcal{M}_f$ with respect to the reward interpolation, and bounding the terms that appear in the reward difference. For convenience, we refer to these offset terms due to the reward as Δ_R. For the final part of this proof, we bound $J(P_{\mathcal{M}_f}, r_{\mathcal{M}})$ in terms of the return on the actual MDP, $J(P_{\mathcal{M}}, r_{\mathcal{M}})$, using the inequality proved above that provides intervals for mixture dynamics but a fixed reward function. Thus, the overall bound is given by $J(\pi, \mathcal{M}_f) \in [J(\pi, \mathcal{M}) - \alpha, J(\pi, \mathcal{M}) + \alpha]$, where α is given by:

$$\alpha = \frac{2\gamma(1-f)}{(1-\gamma)^2} R_{\max} D\left(P_{\mathcal{M}_2}, P_{\mathcal{M}}\right) + \frac{\gamma f}{1-\gamma} \left| \mathbb{E}_{d^{\pi}_{\mathcal{M}} \pi} \left[\left(P^{\pi}_{\mathcal{M}} - P^{\pi}_{\mathcal{M}_1} \right) Q^{\pi}_{\mathcal{M}} \right] \right| + \Delta_R. \quad \text{(C.7.19)}$$

This concludes the proof of this lemma. □

Finally, we prove Theorem 5.2.3 that shows how policy optimization with respect to $\hat{J}(f, \pi)$ affects the performance in the actual MD by using Equation C.7.17 and building on the analysis of pure model-free algorithms from Kumar et al. [181]. We restate a more complete statement of the theorem below and present the constants at the end of the proof.

Theorem C.7.7 (Formal version of Proposition 5.2.3)**.** *Let $\hat{\pi}_{out}(\boldsymbol{a}|\boldsymbol{s})$ be the policy obtained by COMBO. Then, the policy $\pi_{out}(\boldsymbol{a}|\boldsymbol{s})$ is a ζ-safe policy improvement over π_β in the actual MDP*

$\mathcal{M}$, *i.e.*, $J(\pi_{out}, \mathcal{M}) \geq J(\pi_\beta, \mathcal{M}) - \zeta$, *with probability at least* $1-\delta$, *where* ζ *is given by (where* $\rho^\beta(\boldsymbol{s}, \boldsymbol{a}) := d^{\pi_\beta}_{\widehat{\mathcal{M}}}(\boldsymbol{s}, \boldsymbol{a})$*):*

$$\mathcal{O}\left(\frac{\gamma f}{(1-\gamma)^2}\right)\left[\mathbb{E}_{\boldsymbol{s}\sim d^{\pi_{out}}_{\mathcal{M}}}\left[\sqrt{\frac{|\mathcal{A}|}{|\mathcal{D}(\boldsymbol{s})|}(\mathrm{D}_{CQL}(\pi_{out}, \pi_\beta)+1)}\right]\right] + \mathcal{O}\left(\frac{\gamma(1-f)}{(1-\gamma)^2}\right)\mathrm{D}_{\mathrm{TV}}(P_{\mathcal{M}}, P_{\widehat{\mathcal{M}}}) - \beta\frac{\nu(\rho^{\pi_{out}}, f) - \nu(\rho^\beta, f)}{(1-\gamma)}.$$

Proof. We first note that since policy improvement is not being performed in the same MDP, $\mathcal{M}$ as the f-interpolant MDP, $\mathcal{M}_f$, we need to upper and lower bound the amount of improvement occurring in the actual MDP due to the f-interpolant MDP. As a result our first is to relate $J(\pi, \mathcal{M})$ and $J(\pi, \mathcal{M}_f) := J(\overline{\mathcal{M}}, \widehat{\mathcal{M}}, f, \pi)$ for any given policy π.

Step 1: Bounding the return in the actual MDP due to optimization in the f-interpolant MDP. By directly applying Lemma C.7.6 stated and proved previously, we obtain the following upper and lower-bounds on the return of a policy π:

$$J(\overline{\mathcal{M}}, \widehat{\mathcal{M}}, f, \pi) \in \left[J(\pi, \mathcal{M}) - \alpha,\ \ J(\pi, \mathcal{M}) + \alpha\right],$$

where α is shown in Equation C.7.18. As a result, we just need to bound the terms appearing the expression of α to obtain a bound on the return differences. We first note that the terms in the expression for α are of two types: **(1)** terms that depend only on the reward function differences (captured in Δ_R in Equation C.7.19), and **(2)** terms that depend on the dynamics (the other two terms in Equation C.7.19).

To bound Δ_R, we simply appeal to concentration inequalities on reward (Assumption C.7.2), and bound Δ_R as:

$$\begin{aligned}\Delta_R &:= \frac{f}{1-\gamma}\mathbb{E}_{\boldsymbol{s},\boldsymbol{a}\sim d^\pi_{\mathcal{M}}\pi}[|r_{\mathcal{M}_1}(\boldsymbol{s}, \boldsymbol{a}) - r_{\mathcal{M}}(\boldsymbol{s}, \boldsymbol{a})|] + \frac{1-f}{1-\gamma}\mathbb{E}_{\boldsymbol{s},\boldsymbol{a}\sim d^\pi_{\mathcal{M}}\pi}[|r_{\mathcal{M}_2}(\boldsymbol{s}, \boldsymbol{a}) - r_{\mathcal{M}}(\boldsymbol{s}, \boldsymbol{a})|] \\ &\leq \frac{C_{r,\delta}}{1-\gamma}\mathbb{E}_{\boldsymbol{s},\boldsymbol{a}\sim d^\pi_{\mathcal{M}}\pi}\left[\frac{1}{\sqrt{D(\boldsymbol{s}, \boldsymbol{a})}}\right] + \frac{1}{1-\gamma}||R_{\mathcal{M}} - R_{\widehat{\mathcal{M}}}|| := \Delta^u_R.\end{aligned}$$

Note that both of these terms are of the order of $\mathcal{O}(1/(1-\gamma))$ and hence they don't figure in the informal bound in Theorem 5.2.3 in the main text, as these are dominated by terms that grow quadratically with the horizon. To bound the remaining terms in the expression for α, we utilize a result directly from Kumar et al. [181] for the empirical MDP, $\overline{\mathcal{M}}$, which holds for any policy $\pi(\boldsymbol{a}|\boldsymbol{s})$, as shown below.

$$\begin{aligned}&\frac{\gamma}{(1-\gamma)}\left|\mathbb{E}_{\boldsymbol{s},\boldsymbol{a}\sim d^\pi_{\mathcal{M}}(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s})}\left[\left(P^\pi_{\mathcal{M}} - P^\pi_{\mathcal{M}_1}\right)Q^\pi_{\mathcal{M}}\right]\right| \\ &\leq \frac{2\gamma R_{\max} C_{P,\delta}}{(1-\gamma)^2}\mathbb{E}_{\boldsymbol{s}\sim d^\pi_{\mathcal{M}}(\boldsymbol{s})}\left[\frac{\sqrt{|\mathcal{A}|}}{\sqrt{|\mathcal{D}(\boldsymbol{s})|}}\sqrt{D_{\mathrm{CQL}}(\pi, \pi_\beta)(\boldsymbol{s})+1}\right].\end{aligned}$$

Step 2: Incorporate policy improvement in the f-inrerpolant MDP. Now we incorporate the improvement of policy π_{out} over the policy π_β on a weighted mixture of $\widehat{\mathcal{M}}$ and $\overline{\mathcal{M}}$. In what follows, we derive a lower-bound on this improvement by using the fact that policy π_{out} is obtained by maximizing $\hat{J}(f, \pi)$ from Equation C.7.17. As a direct consequence of Equation C.7.17, we note that

$$\hat{J}(f, \pi_{\text{out}}) = J(\overline{\mathcal{M}}, \widehat{\mathcal{M}}, f, \pi_{\text{out}}) - \beta \frac{\nu(\rho^\pi, f)}{1-\gamma} \geq \hat{J}(f, \pi_\beta) = J(\overline{\mathcal{M}}, \widehat{\mathcal{M}}, f, \pi_\beta) - \beta \frac{\nu(\rho^\beta, f)}{1-\gamma} \tag{C.7.20}$$

Following **Step 1**, we will use the upper bound on $J(\overline{\mathcal{M}}, \widehat{\mathcal{M}}, f, \pi)$ for policy $\pi = \pi_{\text{out}}$ and a lower-bound on $J(\overline{\mathcal{M}}, \widehat{\mathcal{M}}, f, \pi)$ for policy $\pi = \pi_\beta$ and obtain the following inequality:

$$\begin{aligned} J(\pi_{\text{out}}, \mathcal{M}) - \beta \frac{\nu(\rho^\pi, f)}{1-\gamma} \geq \Big\{ & J(\pi_\beta, \mathcal{M}) - \beta \frac{\nu(\rho^\beta, f)}{1-\gamma} - \frac{4\gamma(1-f)R_{\max}}{(1-\gamma)^2} D(P_{\mathcal{M}}, P_{\widehat{\mathcal{M}}}) \\ & \underbrace{- \frac{2\gamma f}{(1-\gamma)} \left| \mathbb{E}_{d^{\pi_{\text{out}}}_{\mathcal{M}}} \left[\left(P^{\pi_{\text{out}}}_{\mathcal{M}} - P^{\pi_{\text{out}}}_{\overline{\mathcal{M}}} \right) Q^{\pi_{\text{out}}}_{\mathcal{M}} \right] \right|}_{:=(*)} \\ & \underbrace{- \frac{4\gamma R_{\max} C_{P,\delta} f}{(1-\gamma)^2} \mathbb{E}_{s \sim d^{\pi_\beta}_{\mathcal{M}}} \left[\sqrt{\frac{|\mathcal{A}|}{|\mathcal{D}(s)|}} \right]}_{:=(\wedge)} - \Delta^u_R \Big\}. \end{aligned}$$

The term marked by $(*)$ in the above expression can be upper bounded by the concentration properties of the dynamics as done in Step 1 in this proof:

$$(*) \leq \frac{4\gamma f C_{P,\delta} R_{\max}}{(1-\gamma)^2} \mathbb{E}_{s \sim d^{\pi_{\text{out}}}_{\mathcal{M}}(s)} \left[\frac{\sqrt{|\mathcal{A}|}}{\sqrt{|\mathcal{D}(s)|}} \sqrt{D_{\text{CQL}}(\pi_{\text{out}}, \pi_\beta)(s) + 1} \right]. \tag{C.7.21}$$

Finally, using Equation C.7.21, we can lower-bound the policy return difference as:

$$J(\pi_{\text{out}}, \mathcal{M}) - J(\pi_\beta, \mathcal{M}) \geq \beta \frac{\nu(\rho^\pi, f)}{1-\gamma} - \beta \frac{\nu(\rho^\beta, f)}{1-\gamma} - \frac{4\gamma(1-f)R_{\max}}{(1-\gamma)^2} D(P_{\mathcal{M}}, P_{\widehat{\mathcal{M}}}) - (*) - \Delta^u_R.$$

Plugging the bounds for terms (a), (b) and (c) in the expression for ζ where $J(\pi_{\text{out}}, \mathcal{M}) - J(\pi_\beta, \mathcal{M}) \geq \zeta$, we obtain:

$$\begin{aligned} \zeta = & \left(\frac{4 f \gamma R_{\max} C_{P,\delta}}{(1-\gamma)^2} \right) \mathbb{E}_{s \sim d^{\pi_{\text{out}}}_{\mathcal{M}}(s)} \left[\frac{\sqrt{|\mathcal{A}|}}{\sqrt{|\mathcal{D}(s)|}} \sqrt{D_{\text{CQL}}(\pi_{\text{out}}, \pi_\beta)(s) + 1} \right] + (\wedge) - \Delta^u_R \\ & + \frac{4(1-f)\gamma R_{\max}}{(1-\gamma)^2} D(P_{\mathcal{M}}, P_{\widehat{\mathcal{M}}}) - \beta \frac{\nu(\rho^\pi, f)}{1-\gamma} + \beta \frac{\nu(\rho^\beta, f)}{1-\gamma}. \end{aligned} \tag{C.7.22}$$

□

Remark C.7.8 (Interpretation of Proposition 5.2.3). *Now we will interpret the theoretical expression for ζ in Equation C.7.22, and discuss the scenarios when it is* negative. *When the expression for ζ is negative, the policy π_{out} is an improvement over π_β in the original MDP, $\mathcal{M}$.*

- *First note that we have never used the fact that the learned model $P_{\widehat{\mathcal{M}}}$ is close to the actual MDP, $P_{\mathcal{M}}$ on the states visited by the behavior policy π_β in our analysis. We will use this fact now: in practical scenarios, $v(\rho^\beta, f)$ is expected to be smaller than $v(\rho^\pi, f)$, since $v(\rho^\beta, f)$ is directly controlled by the difference and density ratio of $\rho^\beta(\boldsymbol{s}, \boldsymbol{a})$ and $d(\boldsymbol{s}, \boldsymbol{a})$: $v(\rho^\beta, f) \leq v(\rho^\beta, f = 1) = \sum_{\boldsymbol{s},\boldsymbol{a}} d^{\pi_\beta}_{\widehat{\mathcal{M}}}(\boldsymbol{s}, \boldsymbol{a}) \left(d^{\pi_\beta}_{\widehat{\mathcal{M}}}(\boldsymbol{s}, \boldsymbol{a}) / d^{\pi_\beta}_{\mathcal{M}}(\boldsymbol{s}, \boldsymbol{a}) - 1 \right)^2$ by Lemma C.7.1 which is expected to be small for the behavior policy π_β in cases when the behavior policy marginal in the empirical MDP, $d^{\pi_\beta}_{\mathcal{M}}(\boldsymbol{s}, \boldsymbol{a})$, is broad. This is a direct consequence of the fact that the learned dynamics integrated with the policy under the learned model: $P^{\pi_\beta}_{\widehat{\mathcal{M}}}$ is closer to its counterpart in the empirical MDP: $P^{\pi_\beta}_{\mathcal{M}}$ for π_β. Note that this is not true for any other policy besides the behavior policy that performs several counterfactual actions in a rollout and deviates from the data. For such a learned policy π, we incur an extra error which depends on the importance ratio of policy densities, compounded over the horizon and manifests as the D_{CQL} term (similar to Equation C.7.21, or Lemma D.4.1 in Kumar et al. [181]). Thus, in practice, we argue that we are interested in situations where $v(\rho^\pi, f) > v(\rho^\beta, f)$, in which case by increasing β, we can make the expression for ζ in Equation C.7.22 negative, allowing for policy improvement.*
- *In addition, note that when f is close to 1, the bound reverts to a standard model-free policy improvement bound and when f is close to 0, the bound reverts to a typical model-based policy improvement bound. In scenarios with high sampling error (i.e. smaller $|\mathcal{D}(\boldsymbol{s})|$), if we can learn a good model, i.e., $D(P_{\mathcal{M}}, P_{\widehat{\mathcal{M}}})$ is small, we can attain policy improvement better than model-free methods by relying on the learned model by setting f closer to 0. A similar argument can be made in reverse for handling cases when learning an accurate dynamics model is hard.*

C.8 Experimental details for COMBO

In this section, we include all details of our empirical evaluations of COMBO.

C.8.1 *Practical algorithm implementation details*

Model training. In the setting where the observation space is low-dimensional, as mentioned in Section 5.2.3, we represent the model as a probabilistic neural network that outputs a Gaussian distribution over the next state and reward given the current state and action:

$$\widehat{T}_\theta(\boldsymbol{s}_{t+1}, r | \boldsymbol{s}, \boldsymbol{a}) = \mathcal{N}(\mu_\theta(\boldsymbol{s}_t, \boldsymbol{a}_t), \Sigma_\theta(\boldsymbol{s}_t, \boldsymbol{a}_t)).$$

We train an ensemble of 7 such dynamics models following [147] and pick the best 5 models based on the validation prediction error on a held-out set that contains 1000 transitions in the offline dataset $\mathcal{D}$. During model rollouts, we randomly pick one dynamics model from the best 5 models. Each model in the ensemble is represented as a 4-layer feedforward neural network with 200 hidden units. For the generalization experiments in Section 5.2.5.1, we additionally use a two-head architecture to output the mean and variance after the last hidden layer following [365].

In the image-based setting, we follow Rafailov et al. [272] and use a variational model with the following components:

$$\begin{aligned}
&\text{Image encoder:} && \boldsymbol{h}_t = E_\theta(\boldsymbol{o}_t) \\
&\text{Inference model:} && \boldsymbol{s}_t \sim q_\theta(\boldsymbol{s}_t | \boldsymbol{h}_t, \boldsymbol{s}_{t-1}, \boldsymbol{a}_{t-1}) \\
&\text{Latent transition model:} && \boldsymbol{s}_t \sim \widehat{T}_\theta(\boldsymbol{s}_t | \boldsymbol{s}_{t-1}, \boldsymbol{a}_{t-1}) \\
&\text{Reward predictor:} && r_t \sim p_\theta(r_t | \boldsymbol{s}_t) \\
&\text{Image decoder:} && \boldsymbol{o}_t \sim D_\theta(\boldsymbol{o}_t | \boldsymbol{s}_t).
\end{aligned} \tag{C.8.1}$$

We train the model using the evidence lower bound:

$$\max_\theta \sum_{\tau=0}^{T-1} \left[\mathbb{E}_{q_\theta}[\log D_\theta(\boldsymbol{o}_{\tau+1} | \boldsymbol{s}_{\tau+1})] \right] - \mathbb{E}_{q_\theta} \left[D_{KL}[q_\theta(\boldsymbol{o}_{\tau+1}, \boldsymbol{s}_{\tau+1} | \boldsymbol{s}_\tau, \boldsymbol{a}_\tau) \| \widehat{T}_{\theta_\tau}(\boldsymbol{s}_{\tau+1}, a_{\tau+1})] \right]$$

At each step τ we sample a latent forward model $\widehat{T}_{\theta_\tau}$ from a fixed set of K models $[\widehat{T}_{\theta_1}, \ldots, \widehat{T}_{\theta_K}]$. For the encoder E_θ we use a convolutional neural network with kernel size 4 and stride 2. For the Walker environment we use 4 layers, while the Door Opening task has 5 layers. The D_θ is a transposed convolutional network with stride 2 and kernel sizes $[5, 5, 6, 6]$ and $[5, 5, 5, 6, 6]$ respectively. The inference network has a two-level structure similar to Hafner et al. [127] with a deterministic path using a GRU cell with 256 units and a stochastic path implemented as a conditional diagonal Gaussian with 128 units. We only train an ensemble of stochastic forward models, which are also implemented as conditional diagonal Gaussians.

POLICY OPTIMIZATION. We sample a batch size of 256 transitions for the critic and policy learning. We set $f = 0.5$, which means we sample 50% of the batch of transitions from $\mathcal{D}$ and another 50% from $\mathcal{D}_{\text{model}}$. The equal split between the offline data and the model rollouts strikes the balance between conservatism and generalization in our experiments as shown in our experimental results in Section 5.2.5. We represent the Q-networks and policy as 3-layer feedforward neural networks with 256 hidden units.

For the choice of $\rho(\boldsymbol{s}, \boldsymbol{a})$ in Equation 5.2.1, we can obtain the Q-values that lower-bound the true value of the learned policy π by setting $\rho(\boldsymbol{s}, \boldsymbol{a}) = d^{\pi}_{\widehat{\mathcal{M}}}(\boldsymbol{s})\pi(\boldsymbol{a}|\boldsymbol{s})$. However, as discussed in [181], computing π by alternating the full off-policy evaluation for the policy $\hat{\pi}^k$ at each iteration k and one step of policy improvement is computationally expensive. Instead, following [181], we pick a particular distribution $\psi(\boldsymbol{a}|\boldsymbol{s})$ that approximates the the policy that maximizes the Q-function at the current iteration and set $\rho(\boldsymbol{s}, \boldsymbol{a}) = d^{\pi}_{\widehat{\mathcal{M}}}(\boldsymbol{s})\psi(\boldsymbol{a}|\boldsymbol{s})$. We formulate the new objective as follows:

$$\hat{Q}^{k+1} \leftarrow \arg\min_{Q} \beta \left(\mathbb{E}_{\boldsymbol{s} \sim d^{\pi}_{\widehat{\mathcal{M}}}(\boldsymbol{s}), \boldsymbol{a} \sim \psi(\boldsymbol{a}|\boldsymbol{s})} [Q(\boldsymbol{s}, \boldsymbol{a})] - \mathbb{E}_{\boldsymbol{s}, \boldsymbol{a} \sim \mathcal{D}} [Q(\boldsymbol{s}, \boldsymbol{a})] \right) + \frac{1}{2} \mathbb{E}_{\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}' \sim d_f} \left[\left(Q(\boldsymbol{s}, \boldsymbol{a}) - \widehat{\mathcal{B}}^{\pi} \hat{Q}^k(\boldsymbol{s}, \boldsymbol{a}) \right) \right)^2 \Big] + \mathcal{R}(\psi), \tag{C.8.2}$$

where $\mathcal{R}(\psi)$ is a regularizer on ψ. In practice, we pick $\mathcal{R}(\psi)$ to be the $-D_{\mathrm{KL}}(\psi(\boldsymbol{a}|\boldsymbol{s}) \| \mathrm{Unif}(\boldsymbol{a}))$ and under such a regularization, the first term in Equation C.8.2 corresponds to computing softmax of the Q-values at any state $\boldsymbol{s}$ as follows:

$$\hat{Q}^{k+1} \leftarrow \arg\min_{Q} \max_{\psi} \beta \left(\mathbb{E}_{\boldsymbol{s} \sim d^{\pi}_{\widehat{\mathcal{M}}}(\boldsymbol{s})} \left[\log \sum_{\boldsymbol{a}} Q(\boldsymbol{s}, \boldsymbol{a}) \right] - \mathbb{E}_{\boldsymbol{s}, \boldsymbol{a} \sim \mathcal{D}} [Q(\boldsymbol{s}, \boldsymbol{a})] \right) + \frac{1}{2} \mathbb{E}_{\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}' \sim d_f} \left[\left(Q(\boldsymbol{s}, \boldsymbol{a}) - \widehat{\mathcal{B}}^{\pi} \hat{Q}^k(\boldsymbol{s}, \boldsymbol{a}) \right) \right)^2 \Big]. \tag{C.8.3}$$

We estimate the log-sum-exp term in Equation C.8.3 by sampling 10 actions at every state $\boldsymbol{s}$ in the batch from a uniform policy $\mathrm{Unif}(\boldsymbol{a})$ and the current learned policy $\pi(\boldsymbol{a}|\boldsymbol{s})$ with importance sampling following [181].

C.8.2 *Hyperparameter Selection for COMBO*

In this section, we discuss the hyperparameters that we use for COMBO. In the D4RL and generalization experiments, our method are built upon the implementation of MOPO provided at: https://github.com/tianheyu927/mopo. The hyperparameters used in COMBO that relates to the backbone RL algorithm SAC such as twin Q-functions and number of gradient steps follow from those used in MOPO with the exception of smaller critic and policy learning rates, which we will discuss below. In the image-based domains, COMBO is built upon LOMPO without any changes to the parameters used there. For the evaluation of COMBO, we follow the evaluation protocol in D4RL [92] and a variety of prior offline RL works [181, 365, 166] and report the normalized score of the smooth undiscounted averaged return over 3 random seeds for all environments except sawyer-door-close and sawyer-door where we report the average success rate over 3 random seeds.

We now list the additional hyperparameters as follows.

- **Rollout length h.** We perform a short-horizon model rollouts in COMBO similar to Yu et al. [365] and Rafailov et al. [272]. For the D4RL experiments and generalization

experiments, we followed the defaults used in MOPO and used $h = 1$ for walker2d and sawyer-door-close, $h = 5$ for hopper, halfcheetah and halfcheetah-jump, and $h = 25$ for ant-angle. In the image-based domain we used rollout length of $h = 5$ for both the the walker-walk and sawyer-door-open environments following the same hyperparameters used in Rafailov et al. [272].

- **Q-function and policy learning rates.** On state-based domains, we searched over $\{1e-4, 3e-4\}$ for the Q-function learning rate and $\{1e-5, 3e-5, 1e-4\}$ for the policy learning rate. We found that $3e-4$ for the Q-function learning rate (also used previously in Kumar et al. [181]) and $1e-4$ for the policy learning rate (also recommended previously in Kumar et al. [181] for gym domains) work well for almost all domains except that on walker2d where a smaller Q-function learning rate of $1e-4$ and a correspondingly smaller policy learning rate of $1e-5$ works the best. In the image-based domains, we followed the defaults from prior work [272] and used $3e-4$ for both the policy and Q-function.
- **Conservative coefficient β.** We searched over $\{0.5, 1.0, 5.0\}$ for β, which correspond to low conservatism, medium conservatism and high conservatism. A larger β would be desirable in more narrow dataset distributions with lower-coverage of the state-action space that propagates error in a backup whereas a smaller β is desirable with diverse dataset distributions. On the D4RL experiments, we found that $\beta = 0.5$ works well for halfcheetah agnostic of dataset quality, while on hopper and walker2d, we found that the more "narrow" dataset distributions: medium and medium-expert datasets work best with larger $\beta = 5.0$ whereas more "diverse" dataset distributions: random and medium-replay datasets work best with smaller β ($\beta = 0.5$ for walker2d and $\beta = 1.0$ for hopper) which is consistent with the intuition. On generalization experiments, $\beta = 1.0$ works best for all environments. In the image-domains we use $\beta = 0.5$ for the medium-replay walker-walk task and and $\beta = 1.0$ for all other domains, which again is in accordance with the impact of β on performance.
- **Choice of $\rho(\boldsymbol{s}, \boldsymbol{a})$.** We first decouple $\rho(\boldsymbol{s}, \boldsymbol{a}) = \rho(\boldsymbol{s})\rho(\boldsymbol{a}|\boldsymbol{s})$ for convenience. As discussed in Appendix C.8.1, we use $\rho(\boldsymbol{a}|\boldsymbol{s})$ as the soft-maximum of the Q-values and estimated with log-sum-exp. For $\rho(\boldsymbol{s})$, we searched over $\{d^{\pi}_{\widehat{\mathcal{M}}}, \rho(\boldsymbol{s}) = d_f\}$. We found that $d^{\pi}_{\widehat{\mathcal{M}}}$ works better the hopper task in D4RL while d_f is better for the rest of the environments. For the remaining domains, we found $\rho(\boldsymbol{s}) = d_f$ works well.
- **Choice of $\mu(\boldsymbol{a}|\boldsymbol{s})$.** For the rollout policy μ, we searched $\{\text{Unif}(\boldsymbol{a}), \pi(\boldsymbol{a}|\boldsymbol{s})\}$, i.e. the set that contains a random policy and a current learned policy. We found that $\mu(\boldsymbol{a}|\boldsymbol{s}) = \text{Unif}(\boldsymbol{a})$ works well on the hopper task in D4RL and also in the ant-angle generalization experiment. For the remaining state-based environments, we discovered that $\mu(\boldsymbol{a}|\boldsymbol{s}) = \pi(\boldsymbol{a}|\boldsymbol{s})$ excels. In the image-based domain, we found that $\mu(\boldsymbol{a}|\boldsymbol{s}) = \text{Unif}(\boldsymbol{a})$ works well in the walker-walk domain and $\mu(\boldsymbol{a}|\boldsymbol{s}) = \pi(\boldsymbol{a}|\boldsymbol{s})$ is

better for the sawyer-door environment. We observed that $\mu(\boldsymbol{a}|\boldsymbol{s}) = \text{Unif}(\boldsymbol{a})$ behaves less conservatively and is suitable to tasks where dynamics models can be learned fairly precisely.

- **Choice of Backup.** Following CQL [181], we use the standard deterministic backup for COMBO.
- **Choice of f.** For the ratio between model rollouts and offline data f, we searched $\{0.5, 0.8\}$. We found that $f = 0.8$ works well on the medium and medium-expert in the walker2d task in D4RL. For the remaining tasks, we find $f = 0.5$ works well.

C.8.3 *Automatic Hyperparameter Selection Rule for COMBO*

It is common in prior work on offline RL to select various hyperparameters using online policy rollouts [365, 166, 16, 198]. Requiring online rollouts to tune hyperparameters contradicts the main aim of offline RL, which is to learn entirely from offline data. Therefore, we attempted to devise an automated rule for tuning important hyperparameters such as β and f in a fully offline manner in COMBO. We search over a discrete set of hyperparameters for each task as dicussed above, and use the value of the regularization term $\mathbb{E}_{\boldsymbol{s},\boldsymbol{a}\sim\rho(\boldsymbol{s},\boldsymbol{a})}[Q(\boldsymbol{s},\boldsymbol{a})] - \mathbb{E}_{\boldsymbol{s},\boldsymbol{a}\sim\mathcal{D}}[Q(\boldsymbol{s},\boldsymbol{a})]$ (shown in Eq. 5.2.1) to evaluate the hyperparameters. This automated rule picks the hyperparameter setting which achieves the lowest regularization objective, which indicates that the Q-values on unseen model-predicted state-action tuples are not overestimated.

Below, we provide additional experimental validation showing the effiacy of this automatic hyperparameter selection rule from above. As shown in Table 34, 35, 36 and 37, the above proposed automatic hyperparameter selection rule is able to pick the hyperparameters β, $\mu(\boldsymbol{a}|\boldsymbol{s})$, $\rho(\boldsymbol{s})$ and f and that correspond to the best policy performance based on the regularization value.

C.8.4 *Details of generalization environments*

For halfcheetah-jump and ant-angle, we follow the same environment used in MOPO. For sawyer-door-close, we train the sawyer-door environment in https://github.com/rlworkgroup/metaworld with dense rewards for opening the door until convergence. We collect 50000 transitions with half of the data collected by the final expert policy and a policy that reaches the performance of about half the expert level performance. We relabel the reward such that the reward is 1 when the door is fully closed and 0 otherwise. Hence, the offline RL agent is required to learn the behavior that is different from the behavior policy in a sparse reward setting. We provide the datasets in the following anonymous link[2].

[2] The datasets of the generalization environments are available at the following link: https://drive.google.com/file/d/1pn6dS5OgPQVp_ivGws-tmWdZoU7m_LvC/view?usp=sharing.

Task	$\beta = 0.5$ performance	$\beta = 0.5$ regularizer value	$\beta = 5.0$ performance	$\beta = 5.0$ regularizer value
halfcheetah-medium	**54.2**	**-778.6**	40.8	-236.8
halfcheetah-medium-replay	**55.1**	**28.9**	9.3	283.9
halfcheetah-medium-expert	89.4	189.8	**90.0**	**6.5**
hopper-medium	75.0	-740.7	**97.2**	**-2035.9**
hopper-medium-replay	**89.5**	**37.7**	28.3	107.2
hopper-medium-expert	**111.1**	**-705.6**	75.3	-64.1
walker2d-medium	1.9	51.5	**81.9**	**-1991.2**
walker2d-medium-replay	**56.0**	**-157.9**	27.0	53.6
walker2d-medium-expert	10.3	-788.3	**103.3**	**-3891.4**

Table 34: We include our automatic hyperparameter selection rule of β on a set of representative D4RL environments. We show the policy performance (bold with the higher number) and the regularizer value (bold with the lower number). Lower regularizer value consistently corresponds to the higher policy return, suggesting the effectiveness of our automatic selection rule.

Task	$\mu(\boldsymbol{a}\|\boldsymbol{s}) = \text{Unif}(\boldsymbol{a})$ performance	$\mu(\boldsymbol{a}\|\boldsymbol{s}) = \text{Unif}(\boldsymbol{a})$ regularizer value	$\mu(\boldsymbol{a}\|\boldsymbol{s}) = \pi(\boldsymbol{a}\|\boldsymbol{s})$ performance	$\mu(\boldsymbol{a}\|\boldsymbol{s}) = \pi(\boldsymbol{a}\|\boldsymbol{s})$ regularizer value
hopper-medium	**97.2**	**-2035.9**	52.6	-14.9
walker2d-medium	7.9	-106.8	**81.9**	**-1991.2**

Table 35: We include our automatic hyperparameter selection rule of $\mu(\boldsymbol{a}|\boldsymbol{s})$ on the medium datasets in the hopper and walker2d environments from D4RL. We follow the same convention defined in Table 34 and find that our automatic selection rule can effectively select μ offline.

C.8.5 *Details of image-based environments*

We visualize our image-based environments in Figure 20. We use the standard walker-walk environment from Tassa et al. [319] with 64×64 pixel observations and an action repeat of 2. Datasets were constructed the same way as Fu et al. [92] with 200 trajectories each. For the sawyer-door we use 128×128 pixel observations. The medium-expert dataset contains 1000 rollouts (with a rollout length of 50 steps) covering the state distribution from grasping the door handle to opening the door. The expert dataset contains 1000 trajectories samples from a fully trained (stochastic) policy. The data was obtained from the training process of a stochastic SAC policy using dense reward function as defined in Yu et al. [364]. However, we relabel the rewards, so an agent receives a reward of 1 when the door is fully open and 0 otherwise. This aims to evaluate offline-RL performance in a sparse-reward setting. All the datasets are from [272].

Task	$\rho(s) = d^{\pi}_{\hat{\mathcal{M}}}$ performance	$\rho(s) = d^{\pi}_{\hat{\mathcal{M}}}$ regularizer value	$\rho(s) = d_f$ performance	$\rho(s) = d_f$ regularizer value
hopper-medium	**97.2**	**-2035.9**	56.0	-6.0
walker2d-medium	1.8	14617.4	**81.9**	**-1991.2**

Table 36: We include our automatic hyperparameter selection rule of $\rho(s)$ on the medium datasets in the hopper and walker2d environments from D4RL. We follow the same convention defined in Table 34 and find that our automatic selection rule can effectively select ρ offline.

Task	$f = 0.5$ performance	$f = 0.5$ regularizer value	$f = 0.8$ performance	$f = 0.8$ regularizer value
hopper-medium	**97.2**	**-2035.9**	93.8	-21.3
walker2d-medium	70.9	-1707.0	**81.9**	**-1991.2**

Table 37: We include our automatic hyperparameter selection rule of f on the medium datasets in the hopper and walker2d environments from D4RL. We follow the same convention defined in Table 34 and find that our automatic selection rule can effectively select f offline.

C.9 COMPARING COMBO TO THE NAIVE COMBINATION OF CQL AND MBPO

In this section, we stress the distinction between COMBO and a direct combination of two previous methods CQL and MBPO (denoted as CQL + MBPO). CQL+MBPO performs Q-value regularization using CQL while expanding the offline data with MBPO-style model rollouts. While COMBO utilizes Q-value regularization similar to CQL, the effect is very different. CQL only penalizes the Q-value on unseen actions on the states observed in the dataset whereas COMBO penalizes Q-values on states generated by the learned model while maximizing Q values on state-action tuples in the dataset. Additionally, COMBO also utilizes MBPO-style model rollouts for also augmenting samples for training Q-functions.

To empirically demonstrate the consequences of this distinction, CQL + MBPO performs quite a bit worse than COMBO on generalization experiments (Section 5.2.5.1) as shown in Table 38. The results are averaged across 6 random seeds ($\pm$ denotes 95%-confidence interval of the various runs). This suggests that carefully considering the state distribution, as done in COMBO, is crucial.

Environment	**BatchMean**	**BatchMax**	**COMBO(Ours)**	**CQL+MBPO**
halfcheetah-jump	-1022.6	1808.6	**5392.7**$\pm$575.5	4053.4$\pm$176.9
ant-angle	866.7	2311.9	**2764.8**$\pm$43.6	809.2$\pm$135.4
sawyer-door-close	5%	100%	**100**%$\pm$0.0%	62.7%$\pm$24.8%

Table 38: Comparison between COMBO and CQL+MBPO on tasks that require out-of-distribution generalization. Results are in average returns of halfcheetah-jump and ant-angle and average success rate of sawyer-door-close. All results are averaged over 6 random seeds, $\pm$ the 95%-confidence interval.

Figure 20: Our image-based environments: The observations are 64×64 and 128×128 raw RGB images for the walker-walk and sawyer-door tasks respectively. The sawyer-door-close environment used in in Section 5.2.5.1 also uses the sawyer-door environment.

D

APPENDIX: OFFLINE RL WITH LARGE MODELS

D.1 ADDITIONAL VISUALIZATIONS AND EXPERIMENTS FOR DR3

In this section, we provide visualizations and diagnostic experiments evaluating various aspects of feature co-adaptation and the DR3 regularizer. We first provide more empirical evidence showing the presence of feature co-adaptation in modern deep offline RL algorithms. We will also visualize DR3 inspired from the implicit regularizer term in TD-learning alleviates rank collapse discussed in Kumar et al. [182]. We will compare the efficacies of the explicit regularizer induced for different choices of the noise covariance matrix M (Equation 6.2.3), understand the effect of dropping the stop gradient term in ou practical regularizer and finally, perform diagnostic experiments visualizing if the Q-networks learned with DR3 resemble more like neural networks trained via supervised learning, measured in terms of sensitivity and robustness to layer reinitialization [372].

D.1.1 *More Empirical Evidence of Feature Co-Adaptation*

In this section, we provide more empirical evidence demonstrating the existence of the feature co-adaptation issue in modern offline RL algorithms such as DQN and CQL. As shown below in Figure 21, while the average dataset Q-value for both CQL and DQN exhibit a flatline trend, the dot product similarity for consecutive state-action tuples generally continues to increase throughout training and does not flatline. While DQN eventually diverges in Seaquest, the dot products increase with more gradient steps even before divergence starts to appear.

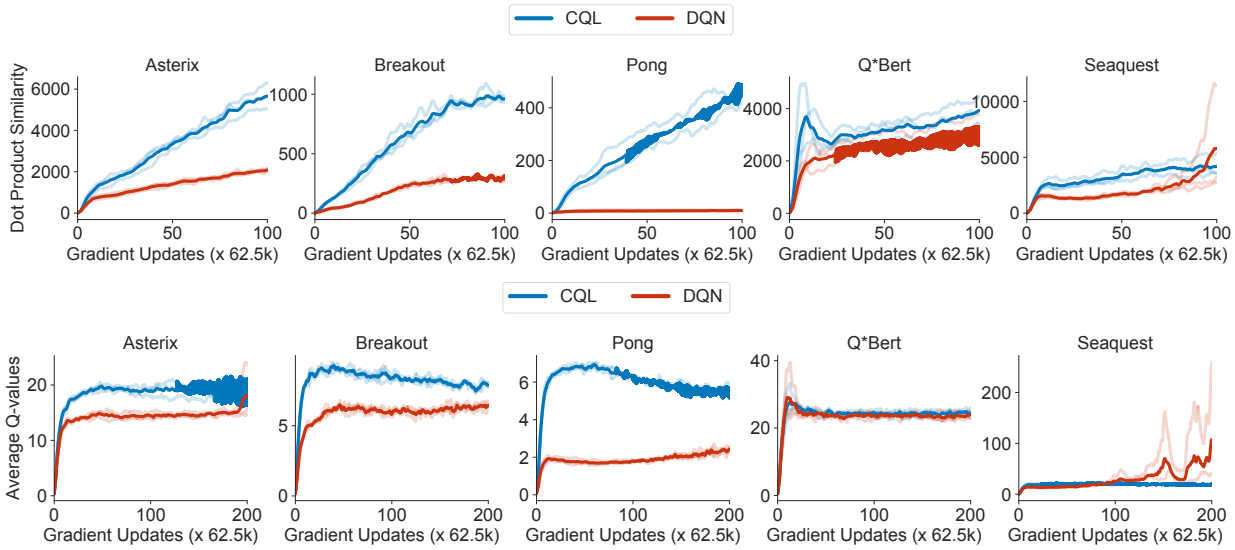


Figure 21: Demonstrating feature co-adaptation on five Atari games with standard offline DQN and CQL, averaged over 3 seeds. Observe that the feature dot products continue to rise with more training for both CQL and DQN, indicating the presence of co-adaptation. On the other hand, the average Q-values exhibit a converged trend, except on Seaquest. Further, note that the dot products continue to increase for CQL even though CQL explicitly corrects for out-of-distribution action inputs.

D.1.2 *Layer-wise structure of a Q-network trained with DR3*

To understand if DR3 indeed makes Q-networks behave as if they were trained via supervised learning, utilizing the empirical analysis tools from Zhang et al. [372], we test the robustness/sensitivity of each layer in the learned network to re-initialization, while keeping the other layers fixed. This tests if a particular layer is *critical* to the predictions of the learned neural network and enables us to reason about generalization properties [372, 40]. We ran CQL and REM and saved all the intermediate checkpoints. Then, as shown in Figure 22, we first loaded a checkpoint (x-axis), and computed policy performance (shaded color; colorbar) by re-initializing a given layer (y-axis) of the network to its initialization value before training for the same run.

Note in Figure 22, that while almost all layers are absolutely critical for the base CQL algorithm, utilizing DR3 substantially reduces sensitivity to the latter layers in the Q-network over the course of training. This is similar to what Zhang et al. [372] observed for supervised learning, where the initial layers of a network were the most critical, and the latter layers primarily performed near-random transformations without affecting the performance of the network. This indicates that utilizing DR3 alters the internal layers of a Q-network trained with TD to behave closer to supervised learning.

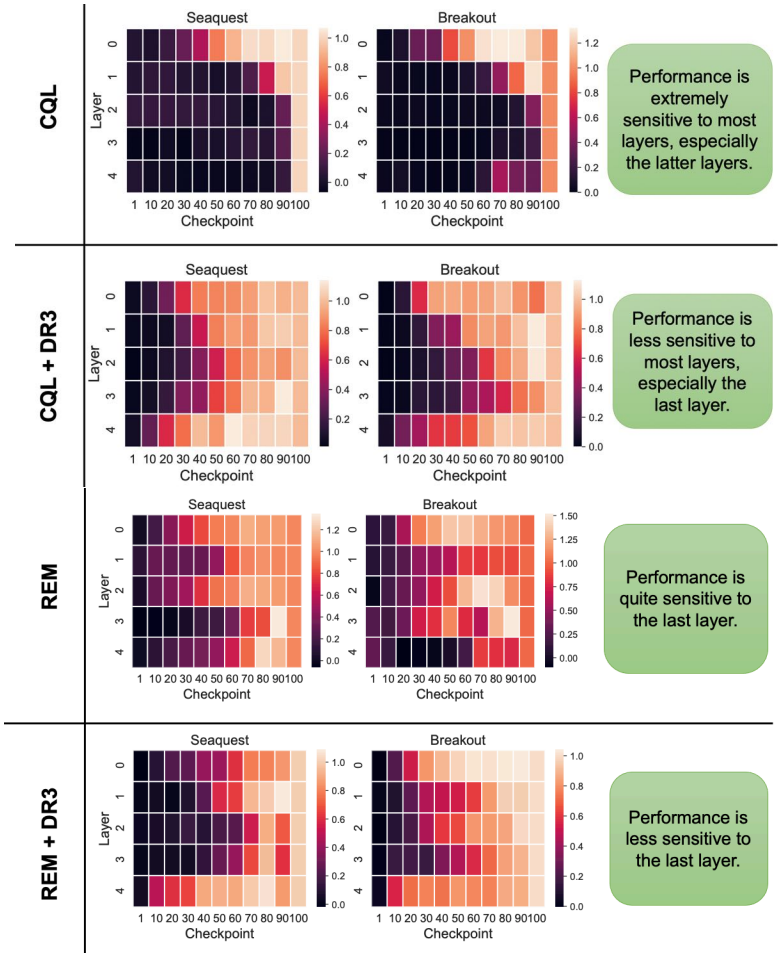


Figure 22: CQL vs CQL + DR3 and REM vs REM + DR3. Average robustness of the learned Q-function to re-initialization of all layers to different checkpoints over the course of training created based on the protocol from Zhang et al. [372]. The colors in the heatmap indicate performance of the reinitialized checkpoint, normalized w.r.t. the checkpoint without any change to layers. Note that while CQL and REM are more sensitive (i.e., less robust) to reinitialization of all the layers especially the last layer, CQL + DR3 and REM + DR3 behave closer to supervised learning, in the sense that they are more robust to reinitialization of layers of the network, especially the last layer.

D.1.3 *Rank Collapse is Alleviated With DR3*

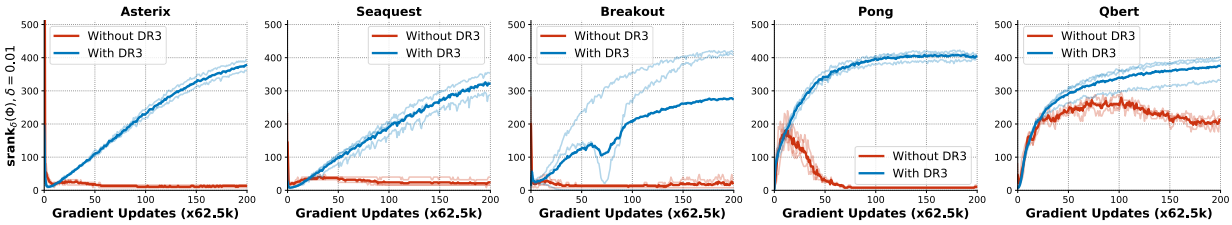


Figure 23: Comparing the feature ranks for CQL and CQL + DR3. Observe that utilizing DR3 successfully alleviates the rank collapse issue noted in prior work without explicitly correcting for it.

Prior work [182] has shown that implicit regularization in TD-learning can lead to a feature rank collapse phenomenon in the Q-function, which hinders the Q-function from using its full representational capacity. Such a phenomenon is absent in supervised learning, where the feature rank does not collapse. Since DR3 is inspired by mitigating the effects of the term in the implicit regularizer (Equation 6.2.3) that only appears in the case of TD-learning, we wish to understand if utilizing DR3 also alleviates rank collapse. To do so, we compute the effective rank $\text{srank}_\delta(\phi)$ metric of the features learned by Q-functions trained via CQL and CQL with DR3 explicit regularizer. As shown in Figure 23, for the case of five Atari games, utilizing DR3 alleviates the rank collapse issue completely (i.e., the ranks do not collapse to very small values when CQL + DR3 is trained for long). We do not claim that the ranks with DR3 are necessarily higher, and infact as we show below, a higher srank of features may not always imply a better solution. The fact that DR3 can prevent rank collapse is potentially surprising, because no term in the practical DR3 regularizer explicitly aims to increase rank: feature dot products can be made smaller while retaining low ranks by simply rescaling the feature vectors. But, as we observe, utilizing DR3 enables learning features that do not exhibit collapsed ranks, thus we hypothesize that correcting for appropriate terms in $R_{TD}(\theta)$ can address some of the previously observed pathologies in TD-learning.

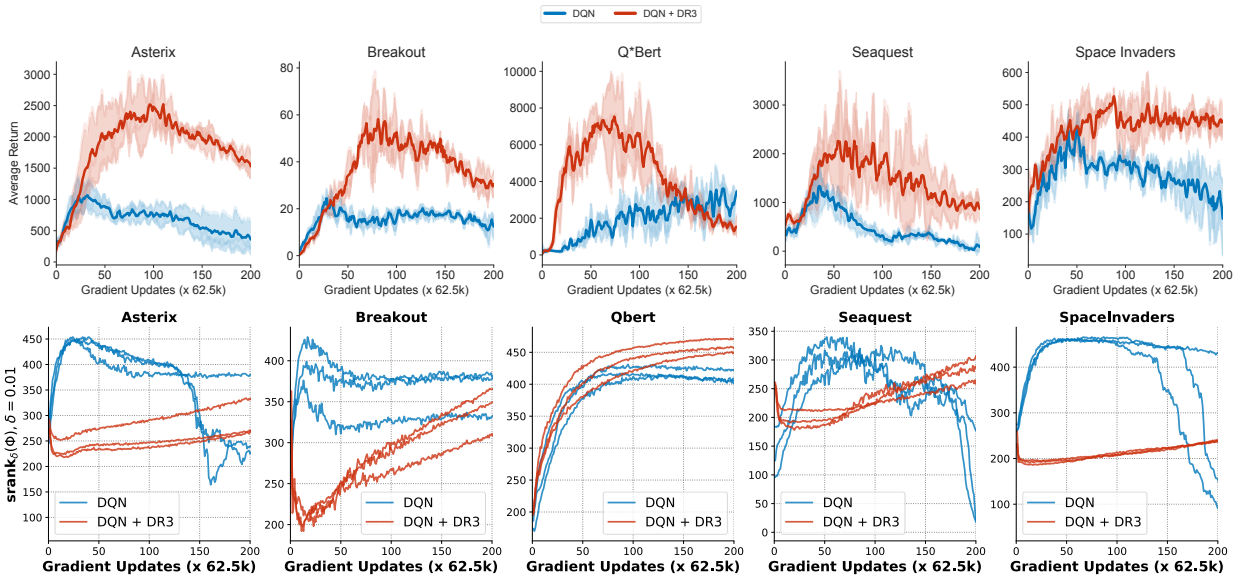


Figure 24: Performance and srank **values for DQN and DQN + DR3.** Observe that the srank values increase for DQN + DR3, while they collapse for DQN on Asterix, Seaquest and SpaceInvaders with more training. Thus, DQN + DR3 does not suffer from a sudden rank collapse. However, a higher srank does not imply a better return, and so while initially DQN does have a high rank, DQN + DR3 performs superiorly.

We now investigate the feature ranks of a Q-network trained when DR3 is applied in conjunction with a standard DQN and REM [6] on the Atari domains. We plot the values of $\text{srank}_\delta(\phi)$, the feature dot products and the performance of the algorithm for DQN in Figure 24 and for REM in Figure 25. In the case of DQN, we find that unlike the base DQN algorithm for which feature rank does begin to collapse with more training, the srank for DQN + DR3 is increasing. We also note that DQN + DR3 attains a better performance compared to DQN, throughout training.

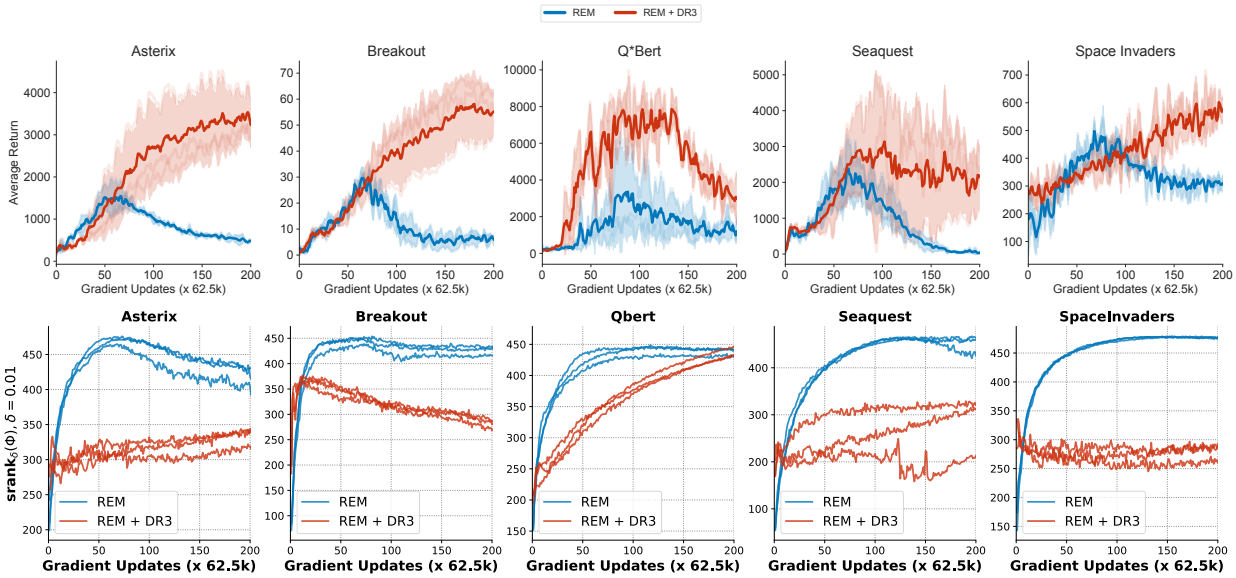


Figure 25: Comparing the performance and srank **values for REM and REM + DR3.** Observe that while REM + DR3 outperforms REM, the srank values attained by REM are much larger than REM + DR3, and none of these ranks have collapsed. Thus, while REM + DR3 maintains non-collapsed features, for the case of REM, it reduces the value of srank and attains better performance. This does not contradict the observations from Kumar et al. [182] as we discuss in the text.

However, we note that the opposite trend is true for the case of REM: while REM + DR3 attains a better performance than REM, adding DR3 leads to a reduction in the srank value compared to base REM. At a first glance, this might seem contradicting Kumar et al. [182], but this is not the case: to our understanding, Kumar et al. [182] establish a correlation between extremely low rank values (i.e., rank collapse) and poor performance, but this does not mean that all high rank features will lead to good performance. We suspect that since REM trains a multi-headed Q-function with shared features and randomized target values, it is able to preserve high-rank features, but this need not mean that these features are useful. In fact, as shown in Figure 26, we find that the base REM algorithm does exhibit feature co-adaptation. This case is an example where the srank metric from Kumar et al. [182] may not indicate poor performance.

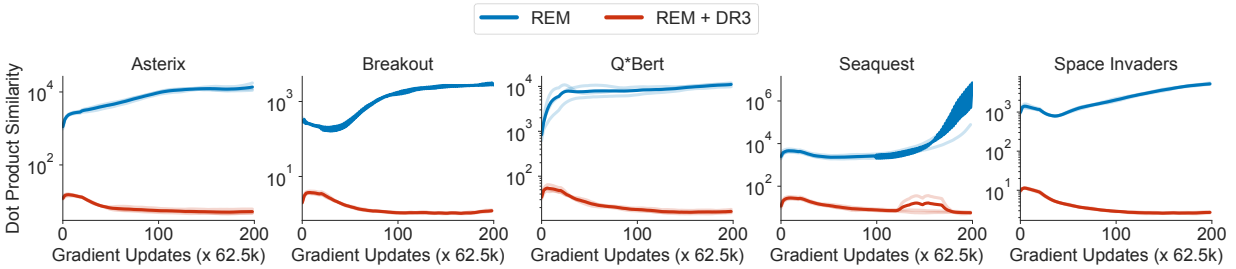


Figure 26: Feature dot products for REM and REM + DR3 on log scale. REM does suffer from feature co-adaptation despite high-rank features.

Table 39: Normalized interquartile mean performance with 95% stratified bootstrap CIs [7] across 17 Atari games of REM, REM + $\Delta'(\Phi)$ (Stop gradient in DR3), REM + DR3 after 6.5M gradient steps for the 1% setting and 12.5M gradient steps for the 5%, 10% settings. Observe that REM + $\Delta'(\phi)$ also improves over the base REM method significantly, by about 130%, even though $\Delta'(\phi)$ is generally comparable and somewhat worse than the DR3 regularizer used in the main paper.

Data	REM	REM + $\Delta'(\Phi)$	REM+DR3
1%	4.0 (3.3, 4.8)	15.0 (13.4, 16.6)	16.5 (14.5, 18.6)
5%	25.9 (23.4, 28.8)	55.5 (50.8, 59.8)	60.2 (55.8, 65.1)
10%	53.3 (51.4, 55.3)	67.7 (64.7, 71.3)	73.8 (69.3, 78)

D.1.4 *Induced Implicit Regularizer: Theory And Practice*

In this section, we compare the performance of our practical DR3 regularizer to the regularizers (Equation 6.2.3) obtained for different choices of M, such as M induced by noise, studied in previous work, and also evaluate the effect of dropping the stop gradient function from the practical version of our regularizer.

Empirically comparing the explicit regularizers for different noise covariance matrices, M. The theoretically derived regularizer (Equation 6.2.3) suggests that for a given choice of M, the following equivalent of feature dot products should increase over the course of training:

$$\Delta_M(\theta) := \sum_{\boldsymbol{s},\boldsymbol{a}\in\mathcal{D}} \text{trace}\left[\Sigma^*_M \nabla Q(\boldsymbol{s},\boldsymbol{a}) \nabla Q(\boldsymbol{s}',\boldsymbol{a}')^\top\right]. \qquad \text{(Generalized dot products)} \quad \text{(D.1.1)}$$

We evaluate the efficacy of the explicit regularizer that penalizes the generalized dot products, $\Delta_M(\theta)$, in improving the performance of the policy, with the goal of identifying if our practical method performs similar to this regularizer on generalized dot products.. While Σ^*_M must be explicitly computed by running fixed point iteration for every parameter iterate θ found during TD-learning – which makes this method significantly computationally expensive[1], we evaluated it on five Atari games for 50 $\times$ 62.5k gradient steps as a proof of concept. As shown in Figure 27, the DR3 penalty with the choice of M which corresponds to label noise, and the dot product DR3 penalty, which is our main practical approach in this paper generally perform similarly on these domains, attaining almost identical learning curves on **4/5 games**, and clearly improving over the base algorithm. This hints at the possibility of utilizing other noise covariance matrices to derive an explicit regularizer. Deriving more computationally efficient versions of the regularizer for a general M and identifying the best choice of M are subject to future work.

[1] In our implementation, we run 20 steps of the fixed-point computation of Σ as shown in Theorem 6.2.1 for each gradient step on the Q-function, and this increases the runtime to about 8 days for 50 iterations on a P100 GPU.

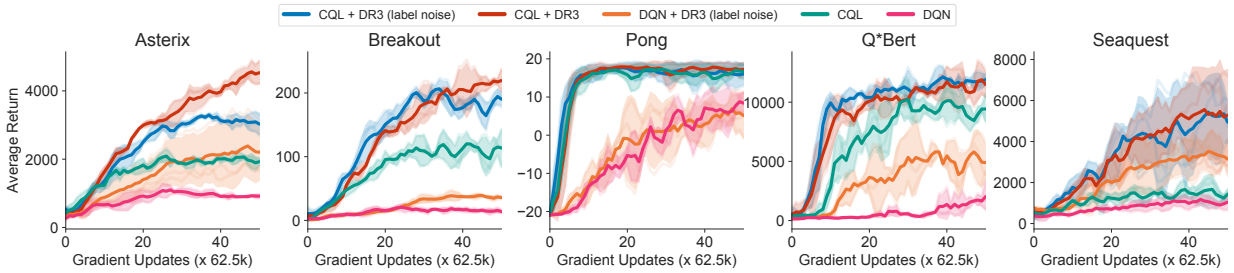


Figure 27: Comparing the performance of explicit penalties for two different choices of the covariance matrix M. Observe that in all the five games the DR3 regularizer derived for the choice of M from Blanc et al. [31] also leads to a substantial increase in performance over the base algorithm, and in four of five games, DR3 (label-noise) works just as well as DR3.

Effect of stop gradient. Finally, we investigate the effect of utilizing a stop gradient in the DR3 regularizer. We run a variant of DR3: $\Delta'(\phi) = \sum_{s,a,s'} \phi(s,a)^\top [[\phi(s',a')]]$, with the stop gradient on the second term (s',a') and present a comparison to the one without the stop gradient in Table 39 for REM as the base offline method, averaged over 17 games. Note that this version of DR3, with the stop gradient, also improves upon the baseline offline RL method (i.e., REM) by **130%**. While this performs largely similar, but somewhat worse than the complete version without the stop gradient, these results do indicate that utilizing $\Delta'(\phi)$ can also lead to significant gains in performance.

D.1.5 *Understanding Feature Co-Adaptation Some More*

In this section, we present some more empirical evidence to understand feature co-adaptation. The three factors we wish to study are: **(1)** the effect of target update frequency on feature co-adaptation; **(2)** understand the trend in normalized similarities and compare these to the trend in dot products; and **(3)** understand the effect of out-on-sample actions in TD-learning and compare it to offline SARSA on a simpler gridworld domain. We answer these questions one by one via experiments aiming to verify each hypothesis.

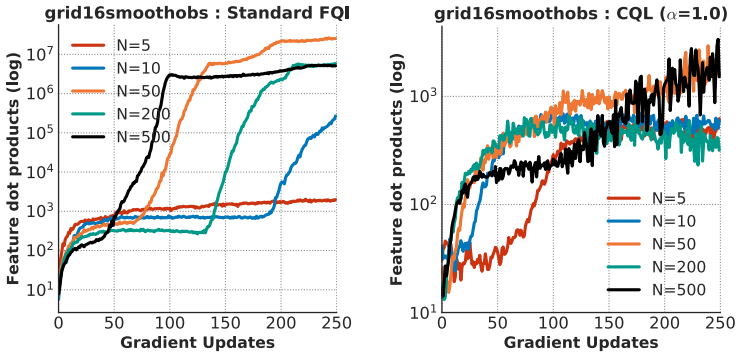


Figure 28: Comparing the feature dot products for various target update delays, where a smaller N implies a faster update and a larger N corresponds to a slower target update. Observe that while slower updates to the target network may reduce co-adaptation, very slow target updates may still lead to excesssive co-adaptation.

D.1.5.1 *Effect of Target Update Frequency on Feature Co-Adaptation*

We studied the effect of target update frequency on feature co-adaptation, on some gridworld domains from Fu et al. [90]. We utilized the grid16smoothobs environment, where the goal of the agent is to navigate from the center of a 16×16 gridworld maze to one of its corners while avoiding obstacles and "lava" cells. The observations provided to the RL algorithm are given by a high-dimensional random transformation of the (x, y) coordinates, smoothed over neighboring cells in the gridworld. We sampled an offline dataset of 256 transitions and trained a Q-network with two hidden layers of size $(1024, 1024)$ via fitted Q-iteration (FQI) [277].

We evaluated the feature dot products for Q-functions trained with a varying target update frequencies, given generically as: updating the target network using a hard target update once per N gradient steps, where N takes on values $N = 5, 10, 50, 100, 200, 500$, and present the results in Figure 28 (left), averaged over 3 random seeds. Thee feature dot products initially decrease from $N = 5$ to $N = 10$, because the target network is updated slower, but then starts to rapidly increase when when the target network is slowed down further to $N = 50$ and $N = 200$ in one case and $N = 500$ in the other case. We also evaluated the feature dot products when using CQL as the base offline RL algorithm. As shown in Figure 28 (right), while CQL does reduce the absolute range of the feature dot products, slow target updates with $N = 500$ still lead to the highest feature dot products as training progresses.

Takeaway: While it is intuitive to think that a slower target network might alleviate co-adaptation, we see that this is not the case empirically with both FQI and CQL, suggesting a deeper question that is an interesting avenue for future study.

D.1.5.2 *Gridworld Experiments Comparing TD-learning And Offline SARSA*

To supplement the analysis in Section 6.2.1, we ran some experiments in the gridworld domains from Fu et al. [90]. In this case, we used the grid16smoothsparse and grid16randomsparse domains, which present challenging navigation tasks in a maze under a 0-1 sparse reward signal, provided at the end of the trajectory. Additionally, the observations available to the offline RL agent do not consist of the raw (x, y) locations of the agent in the maze, but rather high-dimensional randomly chosen transformations of (x, y) in the case of grid16randomsparse, which are additionally smoothed locally around a particular state to obtain grid16smoothsparse.

Since our goal is to compare feature co-adaptation in TD-learning and offline SARSA, we consider a case where we evaluate a "mixed" behavior policy that chooses the optimal action with a probability of 0.7 at a given state, and chooses a random, suboptimal action with 0.3. We then generate a dataset of size 256 transitions and train offline SARSA and TD-learning on this data. While SARSA backups the next action observed in the offline dataset, TD-learning computes a full expectation of the Q-function $\mathbb{E}_{\boldsymbol{a}' \sim \pi_\beta(\cdot|\boldsymbol{s}')}\left[Q(\boldsymbol{s}', \boldsymbol{a}')\right]$ under the behavior policy for computing Bellman backup targets. The behavior policy is fully known to the TD-learning agent. Our Q-network consists of two hidden layers of size $(1024, 1024)$ as before.

We present the trends in the feature dot products for TD-learning and offline SARSA in Figure 29, averaged over three seeds. Observe that the trends in the dot product values for TD-learning and offline SARSA closely follow each other for the initial few gradient steps, soon, the dot products in TD-learning start growing faster. In contrast, the dot products for SARSA either saturate or start decreasing. The only difference between TD-learning and SARSA is the set of actions used to compute Bellman targets – while the actions used for computing Bellman backup targets in SARSA are in-sample actions and are observed in the dataset, the actions used by TD-learning may be out-of-sample, but are still within the distribution of the data-generating behavior policy. This supports our empirical evidence in the main paper showing that out-of-sample actions can lead to feature co-adaptation.

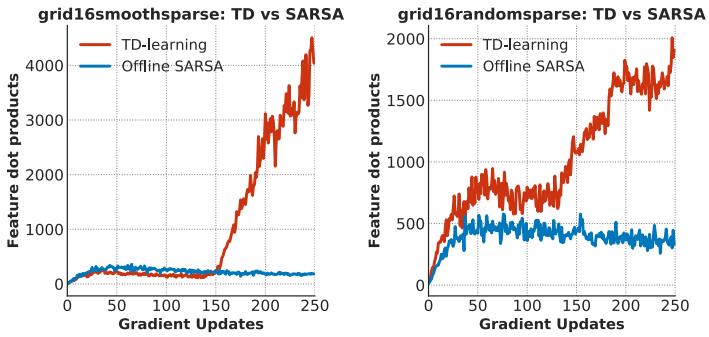


Figure 29: Comparing the feature dot products for TD-learning and offline SARSA, used to compute the value of the behavior policy using a dataset of size 256 on two gridworld domains. Observe that the feature dot products are higher in the case of TD-learning compared to offline SARSA.

D.1.5.3 *Feature Co-Adaptation and Normalized Feature Similarities*

Note that we characterized feature co-adaptation via the dot products of features. In this section, we explore the trends in other notions of similarity, such as cosine similarity between $\phi(\boldsymbol{s}, \boldsymbol{a})$ and $\phi(\boldsymbol{s}', \boldsymbol{a}')$ which measures the dot product of feature vectors at consecutive state-action tuples after normalization. Formally,

$$\cos(\phi(\boldsymbol{s}, \boldsymbol{a}), \phi(\boldsymbol{s}', \boldsymbol{a}')) := \frac{\phi(\boldsymbol{s}, \boldsymbol{a})^\top \phi(\boldsymbol{s}', \boldsymbol{a}')}{||\phi(\boldsymbol{s}, \boldsymbol{a})||_2 \cdot ||\phi(\boldsymbol{s}', \boldsymbol{a}')||_2}.$$

We plot the trend in the cosine similarity with and without DR3 for five Atari games in Figure 30 with CQL, DQN and REM, and for three MuJoCo tasks in Figure 31. We find that the cosine similarity is generally very high on the Atari domains, close to 1, and not indicative of performance degradation. On the Ant and Walker2d MuJoCo domains, we find that the cosine similarity first rises up close to 1 and roughly saturates there. On the Hopper domain, the cosine similarity even decreases over training. However we observe that the feature dot products are increasing for all the domains. Applying DR3 in both cases improves performance (as shown in earlier Appendix), and generally gives rise to reduced cosine similarity values, though it can also increase the cosine similarity values occasionally. Furthermore, even when the cosine similarities were decreasing for the base algorithm (e.g., in the case of Hopper), addition of DR3 reduced the feature dot products and helped improve performance. This indicates that both the norm and directional alignment are contributors to the co-adaptation issue, which is what DR3 aims to fix and independently directional alignment does not indicate poor performance.

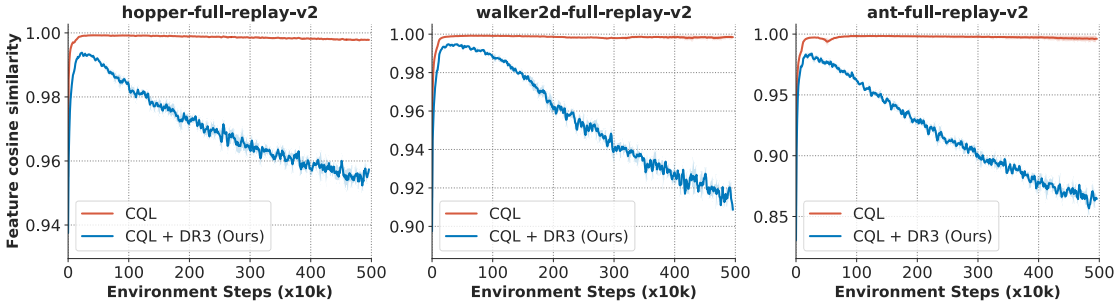


Figure 31: Cosine similarities of CQL and CQL + DR3 on MuJoCo domains. Note that the cosine similarities of CQL grow to 1 and roughly stabilize for Ant and Walker2d, but start decreasing for Hopper, despite the "oscillating" trend of performance on Hopper. This means that cosine similarity need not imply poor performance, and DR3 can improve performance even when cosine similarity of base CQL is decreasing. We also notice that DR3 does actually reduce cosine similarity.

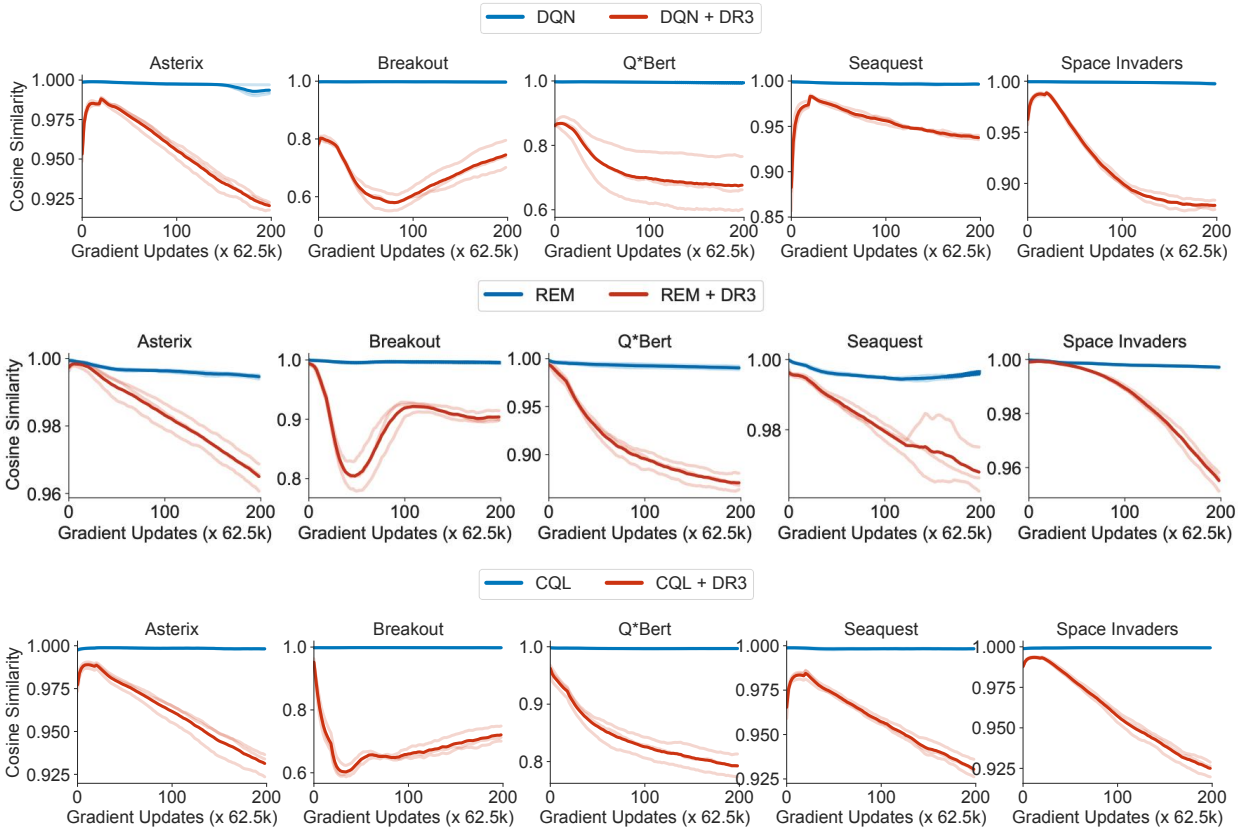


Figure 30: Cosine similarities of DQN, DQN + DR3, REM, REM + DR3 and CQL, CQL + DR3. Note that DQN, REM and CQL attain close to 1 cosine similarities, and addition of DR3 does reduce the cosine similarities of consecutive state-action features.

D.1.6 *Stability of DR3 From a Good Solution*

In this appendix, we study the trend of CQL + DR3 when starting learning from a good initialization, which was studied in Figure 14. As shown in Figure 32, while the performance for baseline CQL degrades significantly (from 5000 at initialization on Asterix, performance degrades to ∼2000 by 100 iterations for base CQL), whereas the performance of DR3 only moves from 5000 to ∼4300. A similar trend holds for Breakout. This means that the addition of DR3 does stabilize the learning relative to the baseline algorithm. Please note that we are not claiming that DR3 is unequivocally stable, but that improves stability relative to the base method.

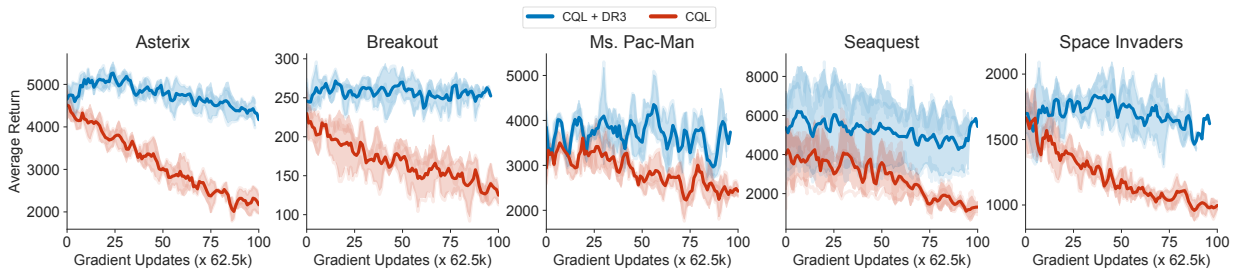


Figure 32: Running CQL + DR3 and CQL in the setup of Figure 14 to evaluate the stability of CQL + DR3 when starting training from a good solution. Observe that the performance of base CQL decays quickly from the good solution, but CQL + DR3 is *relatively* more stable.

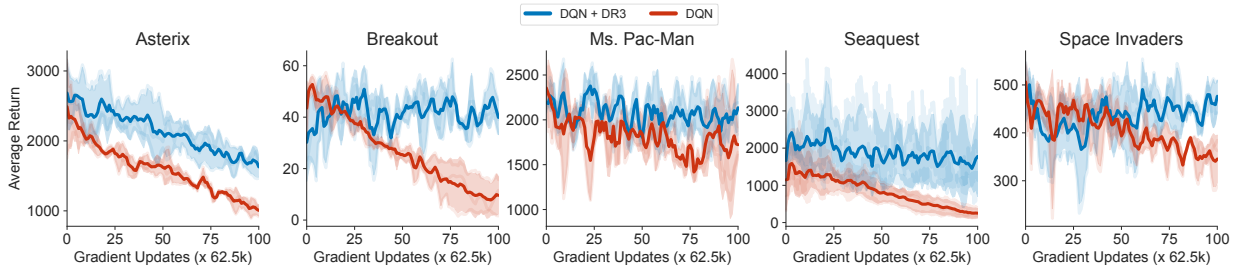


Figure 33: Running DQN + DR3 and DQN in the setup of Figure 14 to evaluate the stability of DQN + DR3 when starting training from a good solution. Observe that the performance of base DQN decays quickly from the good solution, but DQN + DR3 is *relatively* more stable.

D.2 RELATED WORKS

In this section, we briefly review some related works, and in particular, try to connect feature co-adaptation and implicit regularization to various interesting results pertaining to RL lower-bounds with function approximation and self-supervised learning.

D.2.1 *Brief Summary of Related Work*

Prior analyses of the learning dynamics in RL has focused primarily on analyzing error propagation in tabular or linear settings [*e.g.* 43, 68, 349, 335, 336, 79, 56], understanding

instabilities in deep RL [4, 26, 180, 329] and deriving weighted TD updates that enjoy convergence guarantees [224, 225, 315], but these methods do not reason about implicit regularization or any form of representation learning. Ghosh and Bellemare [106] focuses on understanding the stability of TD-learning in underparameterized linear settings, whereas our focus is on the overparameterized setting, when optimizing TD error and learning representations via SGD. Kumar et al. [182] studies the learning dynamics of Q-learning and observes that the rank of the feature matrix, Φ, drops during training. While this observation is related, our analysis characterizes the implicit preference of learning towards feature co-adaptation (Theorem 6.2.1) on out-of-sample actions as the primary culprit for aliasing. Additionally, while the goal of our work is not to increase $\text{srank}(\Phi)$, utilizing Cal-QL not only outperforms the $\text{srank}(\Phi)$ penalty in Kumar et al. [182] by more than **100%**, but it also alleviates rank collapse, with no apparent term that explicitly enforces high rank values. Somewhat related to DR3, Durugkar and Stone [69], Pohlen et al. [263] heuristically constrain gradients of TD to prevent changes in target Q-values to prevent divergence. Contrary to such heuristic approaches, DR3 is inspired from a theoretical model of implicit regularization, and does not prevent changes in target values, but rather reduces feature dot products.

D.2.2 *Extended Related Work*

Lower-bounds for offline RL. Zanette [369] identifies hard instances for offline TD learning of linear value functions when the provided features are "aliased". Note that this work does not consider feature learning or implicit regularization, but their hardness result relies heavily on the fact the given linear features are aliased in a special sense. Aliased features utilized in the hard instance inhibit learning along certain dimensions of the feature space with TD-style updates, necessitating an exponential sample size for near-accurate value estimation, even under strong coverage assumptions. A combination of Zanette [369]'s argument, which provides a hard instance given aliased features, and our analysis, which studies the emergence of co-adapted/similar features in the offline deep RL setting, could imply that the co-adaptation can lead to failure modes from the hard instance, even on standard offline RL problems, when provided with limited data.

Connections to self-supervised learning (SSL). Several modern self-supervised learning methods [115, 45] can be viewwed as utilizing some form of bootstrapping where different augmentations of the same input $(\boldsymbol{x} + \text{Aug}_1, \boldsymbol{x} + \text{Aug}_2)$ serve as consecutive state-action tuples that appear on two sides of the backup. If we may extrapolate our reasoning of feature co-adaptation to this setting, it would suggest that performing noisy updates on a self-supervised bootstrapping loss will give us feature representations that are highly similar for consecutive state-action tuples, i.e., the representations for $\phi(\boldsymbol{x} + \text{Aug}_1)^\top \phi(\boldsymbol{x} + \text{Aug}_2)$ will be high. Intuitively, an easy way for obtaining high feature dot products is for $\phi(\cdot)$ to capture only that information in $\cdot$, which is agnostic to data augmentation, thus giving rise to features that are invariant to transformations. This aligns with what has

been shown in self-supervised learning [323, 324]. Another interesting point to note is that while such an explanation would indicate that highly co-adapted features are beneficial in SSL, such features can be adverse in value-based RL as discussed in Section 6.2.

Preventing divergence in deep TD-learning. Finally, we discuss Achiam et al. [4] which proposes to pre-condition the TD-update using the inverse the neural tangent kernel [145] matrix so that the TD-update is always a contraction, for every θ_k found during TD-learning. Intuitively, this can be overly restrictive in several cases: we do not need to ensure that TD always contracts, but that is eventually stabilizes at good solution over long periods of running noisy TD updates, Our implicit regularizer (Equation'6.2.3) derives this condition, and our theoretically-inspired Cal-QL regularizer shows that empirically, it suffices to penalize the dot product similarity in practice.

D.3 PROOFS FOR DR3

In this section, we will derive our implicit regularizer $R_{\text{TD}}(\theta)$ that emerges when performing TD updates with a stochastic noise model with covariance matrix M. We first introduce our notation that we will use throughout the proof, then present our assumptions and finally derive the regularizer. Our proof utilizes the analysis techniques from Blanc et al. [31] and Damian et al. [52], which analyze label-noise SGD for supervised learning, however key modifications need to be made to their arguments to account for non-symmetric matrices that emerge in TD learning. As a result, the form of the resulting regularizer is very different. To keep the proof concise, we will appeal to lemmas from these prior works which will allow us to bound certain concentration terms.

D.3.1 *Notation*

The noisy TD-learning update for training the Q-function is given by:

$$\theta_{k+1} = \theta_k - \eta \underbrace{\left(\sum_i \nabla_\theta Q(\boldsymbol{s}_i, \boldsymbol{a}_i) \left(Q_\theta(\boldsymbol{s}_i, \boldsymbol{a}_i) - (r_i + \gamma Q_\theta(\boldsymbol{s}'_i, \boldsymbol{a}'_i)) \right) \right)}_{:= g(\theta)} + \eta \varepsilon_k, \quad \varepsilon_k \sim \mathcal{N}(0, M) \tag{D.3.1}$$

where $g(\theta)$ denotes the parameter update. Note that $g(\theta)$ is not a full gradient of a scalar objective, but it is a form of a "pseudo"-gradient or "semi"-gradient. Let ε_k denote an i.i.d.random noise that is added to each update. This noise is sampled from a zero-mean Gaussian random variable with covariance matrix M, i.e., $\mathcal{N}(0, M)$.

Let θ^* denote a point in the parameter space such that in the vicinity of θ^*, $g(\theta) \leq C$, for a small enough C. Let $G(\theta)$ denote the derivative of $g(\theta)$ w.r.t. θ: $G(\theta) = \nabla_\theta g(\theta)$ and let

$\nabla G(\theta)$ denote the third-order tensor $\nabla^2_\theta g(\theta)$. For notation clarity, let $G = G(\theta^*), \nabla G = \nabla G(\theta^*)$. Let e_i denote the signed TD error for a given transition $(\boldsymbol{s}_i, \boldsymbol{a}_i, \boldsymbol{s}'_i) \in \mathcal{D}$ at θ^*:

$$e_i = Q_{\theta^*}(\boldsymbol{s}_i, \boldsymbol{a}_i) - (r_i + \gamma Q_{\theta^*}(\boldsymbol{s}'_i, \boldsymbol{a}'_i)). \tag{D.3.2}$$

Since θ^* is a fixed point of the training TD error, $e_i = 0$. Following Blanc et al. [31], we will assume that the learning rate in gradient descent, η, is small and we will ignore terms that scale as $\mathcal{O}(\eta^{1+\delta})$, for $\delta > 0$. Our proof will rely on using a reference Ornstein-Uhlenbeck (OU) process which the TD parameter iterates will be compared to. Let ζ_k denote the k-th iterate of an OU process, which is defined as:

$$\zeta_{k+1} = (I - \eta G)\zeta_k + \eta \varepsilon_k, \quad \varepsilon_k \sim \mathcal{N}(0, M) \tag{D.3.3}$$

We will drop θ from ∇_θ to indicate that the gradient is being computed at θ^*, and drop $(\boldsymbol{s}_i, \boldsymbol{a}_i)$ from $Q(\boldsymbol{s}_i, \boldsymbol{a}_i)$ and instead represent it as Q_i for brevity; we will represent $Q(\boldsymbol{s}'_i, \boldsymbol{a}'_i)$ as Q'_i. We assume that $\nabla^2 Q_i$ is L_2-Lipschitz and $\nabla^3 Q_i$ is L_3-Lipschitz throughout the parameter space Θ.

D.3.2 *Proof Strategy*

For a given point θ^* to be an attractive fixed point of TD-learning, our proof strategy would be to derive the condition under which it mimics a given OU noise process, which as we will show stays close to the parameter θ^*. This condition would then be interpreted as the gradient of a "induced" implicit regularizer. If the point θ^* is not a stationary point of this regularizer, we will show that the movement θ is large when running the noisy TD updates, indicating that the regularizer, at least in part guides the dynamics of TD-learning. To show this, we would write out the gradient update, isolate some terms that will give rise to the implicit regularizer, and bound the remaining terms using contraction and concentration arguments. The contraction arguments largely follow prior work (though with key exceptions in handling contraction with asymmetric and complex eigenvalue matrices), while the form of the implicit regularizer is different. Finally, we will interpret the resulting update over large timescales to show that learning is indeed guided by the implicit regularizer.

D.3.3 *Assumptions and Conditions*

Next, we present some key assumptions we will need for the proof. Our first assumption is that the matrix $G \in \mathbb{R}^{d \times d}$ is of maximal rank possible, which is equal to the number of datapoints n and $n \ll d$, the dimensionality of the parameter space. Crucially, this assumption do not imply that G is of full rank – it cannot be, because we are in the overparameterized regime.

Assumption D.3.1 (G spans an n-dimensional basis.)**.** *Assume that the matrix G spans n-possible directions in the parameter space and hence, attains the maximal possible rank it can.*

The second condition we require is that the matrices $\sum_i \nabla Q_i \nabla Q_i^\top$ and M share the same n-dimensional basis as matrix G:

Assumption D.3.2. *$\sum_i \nabla Q_i \nabla Q_i^\top$, M, and G span identical n-dimensional subspaces.*

This is a technical condition that is required. If this condition is not met, as we will show the learning dynamics of noisy TD will not be a contraction in certain direction in the parameter space and TD-learning will not stabilize at such a solution θ^*. We will utilize a stronger version of this statement for TD-learning to converge, and we will discuss this shortly.

D.3.4 *Lemmas Used In The Proof*

Next, we present some lemmas that would be useful for proving the theoretical result.

Lemma D.3.3 (Expressions for the first and and second-order derivatives of $g(\theta).$)**.** *The following definitions and expansions apply to our proof:*

$$\begin{aligned} G(\theta^*) &= \sum_i \nabla^2 Q_i e_i + \sum_i \nabla Q_i (\nabla Q_i - \gamma \nabla Q_i')^\top \\ \nabla G(\theta^*)[v, v] &= 2 \sum_i \nabla^2 Q_i v v^\top (\nabla Q_i - \gamma \nabla Q_i') + \sum_i \operatorname{tr}\left((\nabla^2 Q_i - \gamma \nabla^2 Q_i') v v^\top \right) \nabla Q_i \\ &\quad + \nabla^3 Q_i e_i \end{aligned}$$

Lemma D.3.3 presents a decomposition of the matrix G and the directional derivative of the third order tensor $\nabla G[v, v]$ in directions v and v, which will appear in the Taylor expansion layer. Note that at θ^* since $e_i = 0$, the first term in $G(\theta^*)$ and the third term in $\nabla G(\theta^*)[v, v]$ vanish. Lemma D.3.4 derives a fixed-point recursion for the covariance matrix of the total noise accumulated in the OU-process with covariance matrix M and this will appear in our proof.

Lemma D.3.4 (Covariance of the random noise process ζ_k)**.** *Let ζ_k denote the OU process satisfying: $\zeta_{k+1} = (I - \eta G)\zeta_k + \eta \varepsilon_k$, where $\varepsilon_k \sim \mathcal{N}(0, M)$, where $M \succcurlyeq 0$. Then, $\zeta_{k+1} \sim \mathcal{N}(0, \Sigma)$, where Σ satisfies the discrete Lyapunov equation:*

$$\Sigma_M^* = (I - \eta G)\Sigma_M^* (I - \eta G)^\top + \eta^2 M.$$

Proof. For the OU process, $\zeta_{k+1} = (I - \eta G)\zeta_k + \eta \varepsilon_k$, since ε_k is a Gaussian random variable, by induction so is ζ_{k+1}, and therefore the covariance matrix of ζ_{k+1} is given by:

$$\Sigma_{k+1} := (I - \eta G)\Sigma_k (I - \eta G^\top) + \eta^2 M. \tag{D.3.4}$$

Solving for the fixed point for Σ_k gives the desired expression. □

In our proofs, we will require the following contraction lemmas to tightly bound the magnitude of some zero-mean terms that will appear in the noisy TD update under certain scenarios. Unlike the analysis in Damian et al. [52] and Blanc et al. [31] for supervised learning with label noise, where the contraction terms like $(I - \eta G)^k G$ are bounded by $\approx \frac{1}{k\eta}$ intuitively because $I - \eta G$ is a contraction in the subspace spanned by matrix G. However, this is not true for TD-learning directly since terms like $(I - \eta G)^k S$ appear for a different matrix S. Therefore, TD-learning will diverge from θ^* unless matrices G and M have their corresponding eigenvectors assigned to the top eigenvalues be approximately "aligned". We formalize this definition next, and then provide a proof of the concentration guarantee.

Definition D.3.5 ((ω, C_0)-alignment)**.** *Given a positive semidefinite matrix A, let $A = U_A \Lambda_A U_A^\top$ denote its eigendecomposition. Without loss of generality assume that the eiegenvalues are arranged in decreasing order, i.e., $\forall i > j, \Lambda_A(i) \leq \Lambda_A(j)$. Given another matrix B, let $B = U_B \Lambda_B U_B^H$ denote its complex eigendecomposition, where eigenvalues in Λ_B are arranged in decreasing order of their complex magnitudes, i.e., $\forall i > j, |\Lambda_B(i)| \leq |\Lambda_B(j)|$. Then the matrix pair (A, B) is said to be (ω, C_0)-aligned if $|U_B^H(i) U_A(i)| \leq \omega$ and if $\forall\, i, \Lambda_A(i) \leq C_0 |\Lambda_B(i)|$ for a constant C_0.*

If two matrices are (ω, C_0)-aligned, this means that the corresponding eigenvectors when arranged in decreasing order of eigenvalue magnitude roughly align with each other, per the definition of alignment above. This condition would be crucial while deriving the implicit regularizer as it will quantify the rate of contraction of certain terms that define the neighborhood that the iterates of noisy TD-learning will lie in with high probability. We will operate in the setting when the matrix G and $\sum_i \nabla Q_i \nabla Q_i^\top$ are (ω, C_0)-aligned with each other, and matrix M and G are also (ω, C_0)-aligned (note that we can consider ω', C_0'), which will not change our bounds and therefore we go for less notational clutter). Next we utilize this notion of alignment to show a particular contraction bound that extends the weak contraction bound in Damian et al. [52].

Lemma D.3.6. *Assume we are given a matrix G such that $|\lambda_i(I - \eta G)| \leq \rho_0 < 1$ for all λ_i such that $\lambda_i \neq 0$. Let $G = U \Lambda U^H$ be the complex eigenvalue decomposition of G (since almost every matrix is complex-diagonalizable). For a positive semi-definite matrix S that is (ω, C_0)-aligned with G, if $S = U_S \Lambda_S U_S^\top$ is its eigenvalue decomposition, the following contraction bound holds:*

$$||(I - \eta G)^k S|| = \mathcal{O}\left(\frac{\omega C_0}{\eta k}\right)$$

Proof. To prove this statement, we can expand $(I - \eta G)$ using its eigenvalue decomposition only in the subspace that is jointly shared by G and M, and then utilize the definition of ω-alignment to bound the terms.

$$||(I - \eta G)^k S|| = ||(I - \eta U \Lambda U^H)^k U_S \Lambda_S U_S^\top || \tag{D.3.5}$$

$$= \left|\left|(UU^H - \eta U \Lambda_U U^H)^k U_S \Lambda_S U_S^\top \right|\right| \tag{D.3.6}$$

$$= U \left(I - \eta \Lambda\right)^k U^H U_S \Lambda_S U_S^\top \tag{D.3.7}$$

$$\leq \omega \cdot || \left(I - \eta \Lambda\right)^k || \cdot \Lambda_S \tag{D.3.8}$$

$$\leq \omega \cdot C_0 \cdot \left(\max_i \ |1 - \eta \Lambda(i)|^k |\Lambda(i)| \right) \tag{D.3.9}$$

Now we need to solve for the inner maximization term. When $\Lambda(i)$ is not complex for any i, the term above is $\lesssim 1/\eta k$ by applying the result from Damian et al. [52], but when $\Lambda(i)$ is complex, this bound can only hold under certain conditions. To note when this quantity is bounded, we expand $|1 - \eta x|^k$ for some complex number $x = r(\cos\theta + \iota \sin\theta)$:

$$|1 - \eta x|^k = |(1 - \eta r \cos\theta) + \iota \eta r \sin\theta| \tag{D.3.10}$$

$$= \left[\sqrt{(1 - \eta r \cos\theta)^2 + \eta^2 r^2 \sin^2\theta}\right]^k = \left(1 + \eta^2 r^2 - 2\eta r \cos\theta\right)^{k/2} \tag{D.3.11}$$

$$\implies |1 - \eta x|^k |x| = \left(1 + \eta^2 r^2 - 2\eta r \cos\theta\right)^{k/2} r \tag{D.3.12}$$

$$\lesssim \frac{1}{\eta k} \quad \text{if } \eta \leq \min_i \frac{\text{Re}(\Lambda(i))}{|\Lambda(i)|} \quad \text{and} \quad \infty \text{ otherwise.} \tag{D.3.13}$$

Plugging back the above expression in the bound above completes the proof. □

The proof of Lemma D.3.6 indicates that unless the learning rate η and the matrix G are such that the $|\lambda_i(I - \eta G)| \leq \rho < 1$ in directions spanned by matrix S, such an expression may not converge. This is expected since the matrix $I - \eta G$ will not contract in directions of non-zero eigenvalues if the real part $r\cos\theta$ is negative or zero. Additionally, we note that under Definition D.3.5, we can extend several weak-contraction bounds from Damian et al. [52] (Lemmas 9-14 in Damian et al. [52]) to our setting.

Next, Lemma D.3.7 shows that the OU noise iterates are bounded with high probability when Definition D.3.5 holds:

Lemma D.3.7 (ζ_k is bounded with high probability)**.** *With probability atleast* $1 - \delta$ *and under Definition D.3.5,* $||\zeta_k|| \leq n\omega\sqrt{\eta C_0} \log\frac{1}{\delta} = \mathcal{O}(\sqrt{\eta})$.

Proof. To prove this lemma, we first bound the trace of the covariance matrix Σ_{k+1} and then apply high probability bounds on the Martingale norm concentration. The trace

of the covariance matrix Σ_{k+1} can be bounded as follows (all the equations below are restricted to the dimensions of non-zero eigenvalues of G):

$$\text{tr}\left[\Sigma_{k+1}\right] = \sum_{j\le k} \text{tr}\left[(I-\eta G)^j M (I-\eta G^\top)^j\right] \tag{D.3.14}$$

$$= \sum_{j\le k} \text{tr}\left[(UU^H - \eta U\Lambda U^H)^j M (UU^H - \eta U\Lambda U^H)^j\right] \tag{D.3.15}$$

$$= \sum_{j\le k} \text{tr}\left[U(I-\eta\Lambda)^j U^H U_M \Lambda_M U_M^\top U (I-\eta\Lambda)^j U^H\right] \tag{D.3.16}$$

$$= \sum_{j\le k} n\omega^2 C_0 \text{tr}\left[|I-\eta\Lambda|^j \cdot |\Lambda| \cdot |I-\eta\Lambda|^j\right] \tag{D.3.17}$$

$$\le n\omega^2 C_0 \sum_{j\le k} n \cdot \max_\lambda (|1-\eta\lambda|^{2j}\cdot|\lambda|) \le \eta n^2 C_0 \omega^2 \tag{D.3.18}$$

Now, we can apply Corollary 1 from Damian et al. [52] to obtain a bound on $||\zeta_k||$ as with high probability, atleast $1-\delta$, $||\zeta_k|| \le \sqrt{2\text{tr}(\Sigma)\log\frac{1}{\delta}} = n\omega\sqrt{\eta C_0}\log\frac{1}{\delta}$. □

D.3.5 *Main Proof of Theorem 6.2.1*

In this section, we present the main proof of Theorem 6.2.1. The proof involves two components: **(1)** the part where we derive the regularizer, and **(2)** bounding additional terms via concentration inequalities. Part **(1)** is specific to TD-learning, while a lot of the machinery for part **(2)** is directly taken from prior work [52] and Blanc et al. [31]. We focus on part **(1)** here.

Our strategy is to analyze the learning dynamics of noisy TD updates that originate at θ^*. In a small neighborhood around θ^*, we can expand the noisy TD update (Equation 6.2.2) using Taylor's expansion around θ^* which gives:

$$\theta_{k+1} = \theta_k - \eta g(\theta_k) + \eta\varepsilon_k, \quad \varepsilon_k \sim \mathcal{N}(0, M) \tag{D.3.19}$$

$$\implies \theta_{k+1} = \theta_k - \eta\left(g + G(\theta_k-\theta^*) - \frac{\eta}{2}G[\theta_k-\theta^*, \theta_k-\theta^*]\right) + \eta\varepsilon_k + \mathcal{O}(\eta||\theta_k-\theta^*||^3). \tag{D.3.20}$$

Denoting $\nu_k := \theta_k - \theta^*$, using the fact that $||g(\theta^*)|| \le C$, we find that ν_k can be written as:

$$\nu_{k+1} = (I-\eta G)\nu_k + \varepsilon_k + \frac{\eta}{2}G[\nu_k,\nu_k] + \mathcal{O}(\eta||\nu_k||^3 + \eta C) \tag{D.3.21}$$

Since the OU process ζ_k stays in the vicinity of the point θ^*, and follows a similar recursion to the one above, our goal would be to design a regularizer so that Equation D.3.21 closely

follows the OU process. Thus, we would want to bound the difference between the variable ν_k and the variable ζ_k, denoted as r_k to be within a small neighborhood:

$$r_{k+1} = \nu_{k+1} - \zeta_{k+1} = (I - \eta G)(\nu_k - \zeta_k)_{r_k} + \frac{1}{2} G[\nu_k, \nu_k] + \mathcal{O}(\eta ||\nu_k||^3 + \eta C).$$

We can write down an expression for r_k summing over all the terms:

$$r_{k+1} = \underbrace{-\frac{\eta}{2} \sum_{j \le k} (I - \eta G)^{k-j} \nabla G[\nu_k, \nu_k]}_{\text{term (a)}} + \underbrace{\sum_{j \le k} (I - \eta G)^j \left[\mathcal{O}(\eta ||\nu_k||^3 + \eta C) \right]}_{\text{term (b)}}. \tag{D.3.22}$$

Term (a) in the above equation is the one that can induce a displacement in r_k as k increases and would be used to derive the regularizer, whereas term (b) primarily consists of terms that concentrate to 0. We first analyze term (a) and then we will analyze the concentration terms later.

To analyze term (a), note that the term $\nabla G[\nu_k, \nu_k]$, by Lemma D.3.3, only depends on ν_k via the covariance matrix $\nu_k \nu_k^\top$. So we will partition this term into two terms: **(i)** a term that utilizes the asymptotic covariance matrix of the OU process and **(ii)** errors due to a finite k and stochasticity that will concentrate.

$$2 \times \text{(a)} = \eta \sum_{j \le k} (I - \eta G)^{k-j} \nabla G[\nu_k, \nu_k] \tag{D.3.23}$$

$$= \sum_{j \le k} (I - \eta G)^{k-j} \nabla G[\zeta^*, \zeta^*] + \sum_{j \le k} (I - \eta G)^{k-j} \nabla G([\nu_k, \nu_k] - [\zeta^*, \zeta^*]), \tag{D.3.24}$$

The first term is a "bias" term and doesn't concentrate to 0, and will give rise to the regularizer. We can break this term using Lemma D.3.3 as:

$$\nabla G[\zeta^*, \zeta^*] = 2 \sum_i \nabla^2 Q_i \Sigma_M^* (\nabla Q_i - \gamma \nabla Q_i') + \sum_i \text{tr} \left[(\nabla^2 Q_i - \gamma \nabla^2 Q_i') \Sigma_M^* \right] \nabla Q_i \tag{D.3.25}$$

The regularizer $R_{\text{TD}}(\theta)$ is the function such that:

$$\nabla_\theta R_{\text{TD}}(\theta) = \sum_i \nabla^2 Q_i \Sigma_M^* (\nabla Q_i - \gamma \nabla Q_i') \tag{D.3.26}$$

$$\implies R_{\text{TD}}(\theta) = \sum_i \nabla Q_i \Sigma_M^* \nabla Q_i^\top - \gamma \sum_i \text{trace} \left(\Sigma_M^* \nabla Q_i [[\nabla Q_i']]^\top \right), \tag{D.3.27}$$

where $[[\cdot]]$ denotes the stop gradient operator. If the point θ^* is a stationary point of the regularizer $R_{\text{TD}}(\theta)$, then Equations D.3.26 and D.3.27 imply that the first term of Equation D.3.25 must be 0. Therefore in this case to show that θ^* is attractive, we need to show that the other terms in Equations D.3.25, D.3.24 and term (b) in Equation D.3.22 concentrate around 0 and are bounded in magnitude. The remaining part of the proof shown in Appendix D.3.7 provides these details, but we first summarize the main takeaways in the proof to conclude the argument.

D.3.6 *Summary of the Argument*

We will show how to concentrate terms in Equation D.3.26 besides the regularizer largely following the techniques from prior work, but we first summarize the entire proof. The overall update to the vector r_k which measures the displacement between the parameter vector $\theta_k - \theta^*$ and the OU-process ζ_k can be written as follows, and it is governed by the derivative of the implicit regularizer (modulo error terms):

$$r_{k+1} = -\frac{\eta}{2}\sum_{j\le k}(I-\eta G)^{k-j}\nabla_\theta R_{\text{TD}}(\theta^*) + \mathcal{O}\left(\sqrt{\eta t}\cdot \text{poly}(C, L_2, L_3, \omega, C_0)\right). \tag{D.3.28}$$

An important detail to note here is that since the regularizer consists of Σ_M^* and the size of Σ_M^* (i.e, its eigenvalues), as shown in Lemma D.3.7 depends on one factor of η. So, effectively the first term in Equation D.3.28 does depend on two factors of η. Using Equation D.3.28, we can write the deviation between θ^* and θ_k as:

$$\nu_{k+1} = \zeta_{k+1} - \frac{\eta}{2}\sum_{j\le k}(I-\eta G)^{k-j}\nabla_\theta R_{\text{TD}}(\theta^*) + \mathcal{O}\left(\sqrt{\eta t}\cdot \text{poly}(C, L_2, L_3, \omega, C_0)\right). \tag{D.3.29}$$

The OU process ζ_k converges to θ^* in the subspace spanned by G, since the condition $\rho(I-\eta G) < 1$ is active in this subspace (if the condition that $\rho(I-\eta G) < 1$ in the subspace spanned by G is not true, then as Ghosh and Bellemare [106] show, TD can diverge). Now, given G satisfies this spectral radius condition, ζ_k would converge to θ^* within a timescale of $\mathcal{O}\left(\frac{1}{\eta}\right)$ within this subspace, which as Blanc et al. [31] put it is the strength of the "mean-reversion" term. On the remaining directions (note that $d \gg n$), the dynamics is guided by the regularizer, although with a smaller weight of η^2.

D.3.7 *Additional Proof Details: Concentrating Other Terms*

We first concentrate the terms in Equation D.3.25. The cumulative effect of the second term in Equation D.3.25 is given by:

$$\eta\sum_{j\le k}(I-\eta G)^{j-k}\nabla Q_i \text{tr}\left[(\nabla^2 Q_i - \gamma\nabla^2 Q_i')\Sigma_M^*\right] \tag{D.3.30}$$

$$\le \eta\sum_{j\le k}(I-\eta G)^{j-k}\nabla Q_i\cdot\mathcal{O}\left(L_2(1+\gamma)\sigma\right) \le \mathcal{O}\left(\eta\sqrt{\frac{k}{\eta}}\omega_0 C_0 L_2(1+\gamma)\sigma\right), \tag{D.3.31}$$

which follows from the fact that $\nabla^2 Q_i$ is L_2-Lipschitz, and using Lemma D.3.6 for contracting the remaining terms.

Next, we turn to concentrating the second term in Equation D.3.24. This term corresponds to the contribution of difference between the empirical covariance matrix $\nu_k\nu_k^\top$ and the

asymptotic covariance matrix $\zeta^*\zeta^{*\top}$. We expand this term below using the form of G from Lemma D.3.3, and bound it one by one.

$$\sum_{j\le k}(I-\eta G)^{k-j}\nabla G([\nu_k,\nu_k]-[\zeta^*,\zeta^*]) \tag{D.3.32}$$

$$=\sum_{j\le k}\sum_{i}(I-\eta G)^{k-j}\nabla^2 Q_i\left(\nu_k\nu_k-\zeta^*\zeta^{*\top}\right)(\nabla Q_i-\gamma\nabla Q_i')+\mathcal{O}\left(\sqrt{\eta k}\omega_0 C_0 L_2(1+\gamma)\sigma\right) \tag{D.3.33}$$

Now, we note that the term $\Delta_{k+1}:=\nu_{k+1}\nu_{k+1}^\top-\zeta^*\zeta^{*\top}$ can itself be written as a recursion:

$$\Delta_{k+1}=(I-\eta G)(\Delta_k)(I-\eta G)^\top+(I-\eta G)\zeta_k\varepsilon^\top+\varepsilon\zeta_k^\top(I-\eta G)^\top{}_{A_k}+\varepsilon\varepsilon^\top-\eta M_{B_k} \tag{D.3.34}$$

Expanding the term Δ_{k+1} in terms of a summation over k, and plugging it into the expression from Equation D.3.33 we get

$$\sum_i\sum_{j\le k}(I-\eta G)^{k-j}\nabla^2 Q_i(I-\eta G)^j\Delta_0(I-\eta G^\top)^j \tag{D.3.35}$$
$$+\sum_i\sum_{j\le k}\sum_{p\le j}(I-\eta G)^{k-j}\nabla^2 Q_i(I-\eta G)^{j-p-1}(A_p+B_p)(I-\eta G^\top)^{j-p-1}$$

Now by noting that if G and ∇Q_i are (ω,C_0)-aligned, then so are $G^\top$ and ∇Q_i, we can finish the proof by repeating the calculations used by Damian et al. [52] (Appendix B, Equations 67-73) to bound the terms in Equation D.3.35 by $\mathcal{O}(\sqrt{\eta k})$, but with an additional factor of $\omega^2C_0^2$.

Term (b) in Equation D.3.22. When C is small enough, we can bound the term (b) using $\mathcal{O}(\sqrt{\eta k})$, similar to Damian et al. [52].

D.4 Proof of Proposition 6.2.2

In this section, we will prove Proposition 6.2.2. First, we refer to Proposition 3.1 in Ghosh and Bellemare [106], which shows that TD-learning is stable and converges if and only if the matrix $M_\phi=\Phi^\top(\Phi-\gamma\Phi')$ has eigenvalues with all positive real entries. Now note that if,

$$\sum_{s,a}\phi(s,a)^\top\phi(s,a)\le\gamma\sum_{s,a,s'}\phi(s',a')^\top\phi(s,a) \tag{D.4.1}$$

$$\implies \text{trace}\left(\Phi^\top\Phi\right)\le\gamma\text{trace}\left(\Phi^\top\Phi'\right) \tag{D.4.2}$$

$$\implies \text{trace}\left[\Phi^\top\left(\Phi-\gamma\Phi'\right)\right]\le 0. \tag{D.4.3}$$

Since the trace of a real matrix is the sum of real components of eigenvalues, if for a given matrix M, $\text{trace}(M) \leq 0$, then there exists atleast one eigenvalue λ_i such that $\text{Re}(\lambda_i) \leq 0$. If $\lambda_i < 0$, then the learning dynamics of TD would diverge, while if $\lambda_i = 0$ for all i, then learning will not contract towards the TD fixed point. This concludes the proof of this result.

D.5 EXPERIMENTAL DETAILS OF APPLYING DR3

In this section, we discuss the practical experimental details and hyperparameters in applying our method, DR3 to various offline RL methods. We first discuss an overview of the offline RL methods we considered in this paper, and then provide a discussion of hyperparameters for DR3.

D.5.1 *Background on Various Offline RL Algorithms*

In this paper, we consider four base offline RL algorithms that we apply DR3 on. These methods are detailed below:

REM. Random ensemble mixture [6] is an uncertainty-based offline RL algorithm uses multiple parameterized Q-functions to estimate the Q-values. During the Bellman backup, REM computes a random convex combination of the target Q-values and then trains the Q-function to match this randomized target estimate. The randomized target value estimate provides a robust estimate of target values, and delays unlearning and performance degradation that we typically see with standard DQN-style algorithms in the offline setting. For instantiating REM, we follow the instantiation provided by the authors and instantiate a multi-headed Q-function with 200 heads, each of which serves as an estimate of the target value. These multiple heads branch off the last-but-one layer features of the base Q-network. The objective for REM is given by:

$$\min_{\theta} \mathbb{E}_{\boldsymbol{s},\boldsymbol{a},r,\boldsymbol{s}' \sim \mathcal{D}} \left[\mathbb{E}_{\alpha_1,\ldots,\alpha_K \sim \Delta} \left[\left(\sum_k \alpha_k^k(\boldsymbol{s},\boldsymbol{a}) - r - \gamma \max_{\boldsymbol{a}'} \sum_k \alpha_k^k(\boldsymbol{s}',\boldsymbol{a}') \right) \right] \right] \tag{D.5.1}$$

where l_λ denotes the Huber loss while P_Δ denotes the probability distribution over the standard (K 1)-simplex.

CQL. Conservative Q-learning [181] is an offline RL algorithm that learns a conservative value function such that the estimated performance of the policy under this learned value function lower-bounds its true value. CQL modifies the Q-function training to incorporate a term that minimizes the overestimated Q-values in expectation, while maximizing the Q-values observed in the dataset, in addition to standard TD error. This CQL regularizer is typically multiplied by a coefficient α, and we pick $\alpha = 0.1$ for all our Atari experiments following Kumar et al. [182] and $\alpha = 5.0$ for all our kitchen and antmaze D4RL experiments. Using $\overline{y}_k(\boldsymbol{s},\boldsymbol{a})$ to denote the target values computed via

the Bellman backup (we use actor-critic backup for D4RL experiments and the $\max_{a'}$ backup for standard Q-learning in our Atari experiments following Kumar et al. [181]), the objective for training CQL is given by:

$$\alpha \left(\mathbb{E}_{s \sim \mathcal{D}} \left[\log \sum_{a} \exp(Q(s, a)) \right] - \mathbb{E}_{s,a \sim \mathcal{D}} \left[Q(s, a) \right] \right) + \frac{1}{2} \mathbb{E}_{s,a,s' \sim \mathcal{D}} \left[\left(Q(s, a) - \overline{y}_k(s, a) \right)^2 \right].$$

For Atari, we utilize the standard convolutional neural network from Agarwal et al. [6], Kumar et al. [182] with 3 convolutional layers borrowed from the nature DQN network and then a hidden feedforward layer of size 512.

COG. COG [303] is an algorithmic framework for utilizing large, unlabeled datasets of diverse behavior to learn generalizable policies via offline RL. Similar to real-world scenarios where large unlabeled datasets are available alongside limited task-specific data, the agent is provided with two types of datasets. The task-specific dataset consists of behavior relevant for the task, but the prior dataset can consist of a number of random or scripted behaviors being executed in the same environment/setting. The goal in this task is to actually stitch together relevant and overlapping parts of different trajectories to obtain a good policy that can work from a new initial condition that was not seen in a trajectory that actually achieved the reward. COG utilizes CQL as the base offline RL algorithm, and following Singh et al. [303], we fix the hyperparameter $\alpha = 1.0$ in the CQL part for both base COG and COG + DR3. All other hyperparameters including network sizes, etc are kept fixed as the prior work Singh et al. [303] as well.

D.5.2 *Tasks and Environments Used*

Atari 2600 games used. For all our experiments, we used the same set of 17 games utilized by Kumar et al. [182] to test rank collapse. In the case of Atari, we used the 5 standard games (Asterix, Qbert, Pong, Seaquest, Breakout) for tuning the hyperparameters, a strategy followed by several prior works [116, 6, 182]. The 17 games we test on are: Asterix, Qbert, Pong, Seaquest, Breakout, Double Dunk, James Bond, Ms. Pacman, Space Invaders, Zaxxon, Wizard of Wor, Yars' Revenge, Enduro, Road Runner, BeamRider, Demon Attack, Ice Hockey.

Following Agarwal et al. [7], we report interquartile mean (IQM) normalized scores across all runs as mean scores can be dominated by performance on a few outlier tasks while median is independent of performance on all except 1 task – zero score on half of the tasks would not affect the median. IQM which corresponds to 25% trimmed mean and considers the performance on middle 50% of the runs. IQM interpolates between mean and median, which correspond to 0% and almost 50% trimmed means across runs.

Robotic manipulation tasks from COG [303]. These tasks consist of a 6-DoF WidowX robot, placed in front of two drawers and a larger variety of objects. The robot can open or close a drawer, grasp objects from inside the drawer or on the table, and place them

Table 40: Hyperparameters used by the offline RL Atari agents in our experiments. Following Agarwal et al. [6], the Atari environments used by us are stochastic due to sticky actions, *i.e.* there is a 25% chance at every time step that the environment will execute the agents previous action again, instead of the new action commanded. We report offline training results with same hyperparameters over 5 random seeds of the offline dataset, game simulator and network initialization.

Hyperparameter	Setting (for both variations)
Sticky actions	Yes
Sticky action probability	0.25
Grey-scaling	True
Observation down-sampling	(84, 84)
Frames stacked	4
Frame skip (Action repetitions)	4
Reward clipping	[-1, 1]
Terminal condition	Game Over
Max frames per episode	108K
Discount factor	0.99
Mini-batch size	32
Target network update period	every 2000 updates
Training environment steps per iteration	250K
Update period every	4 environment steps
Evaluation ϵ	0.001
Evaluation steps per iteration	125K
Q-network: channels	32, 64, 64
Q-network: filter size	8×8, 4×4, 3×3
Q-network: stride	4, 2, 1
Q-network: hidden units	512

anywhere in the scene. The task here consists of taking an object out of a drawer. A reward of +1 is obtained when the object has been taken out, and zero otherwise. There are two variants of this domain: **(1)** in the first variant, the drawer starts out closed, the top drawer starts out open (which blocks the handle for the lower drawer), and an object starts out in front of the closed drawer, which must be moved out of the way before opening, and **(2)** in the second variant, the drawer is blocked by an object, and this object must be removed before the drawer can be opened and the target object can be grasped from the drawer. The prior data for this environment is collected from a collection of scripted randomized policies. These policies are capable of opening and closing both drawers with 40-50% success rates, can grasp objects in the scene with about a 70% success rate, and place those objects at random places in the scene (with a slight bias for putting them in the tray).

D.5.3 *The DR3 Regularizer Coefficient*

We utilize identical hyperparameters of the base offline RL algorithms when DR3 is used, where the base hyper-parameters correspond to the ones provided in the corresponding publications. DR3 requires us to tune the additional coefficient c_0, that weights the DR3 explicit regularizer term. In order to find this value on our domains, we followed the tuning strategy typically followed on Atari, where we evaluated four different values of $c_0 \in \{0.001, 0.01, 0.03, 0.3\}$ on 5 games (Asterix, Seaquest, Breakout, Pong and SpaceInvaders) on the 5% replay dataset settings, picked c_0 that wprked best on just these domains, and used it to report performance on all 17 games, across all dataset settings (1% replay and 10% initial replay) in Section 6.2.4. This protocol is standard in Atari and has been used previously in Agarwal et al. [6], Gulcehre et al. [116], Kumar et al. [182] in the context of offline RL. The value of the coefficient found using this strategy was $c_0 = 0.001$ for REM and $c_0 = 0.03$ for CQL.

D.6 Full results for DR3

Table 41: Normalized interquartile mean (IQM) final performance (last iteration return) of CQL, CQL + DR3, REM and REM + DR3 after 6.5M gradient steps for the 1% setting and 12.5M gradient steps for the 5%, 10% settings. Intervals in brackets show 95% CIs computed using stratified percentile bootstrap [7]

Data	CQL	CQL + DR3	REM	REM + DR3
1%	44.4 (31.0, 54.3)	**61.6** (39.1, 71.5)	0.0 (-0.7, 0.1)	**13.1** (9.9, 18.3)
5%	89.6 (67.9, 98.1)	**100.2** (90.6, 102.7)	3.9 (3.1, 7.6)	**74.8** (59.6, 84.4)
10%	57.4 (53.2, 62.4)	**67.0** (62.8, 73.0)	24.9 (15.0, 29.1)	**72.4** (65.7, 81.7)

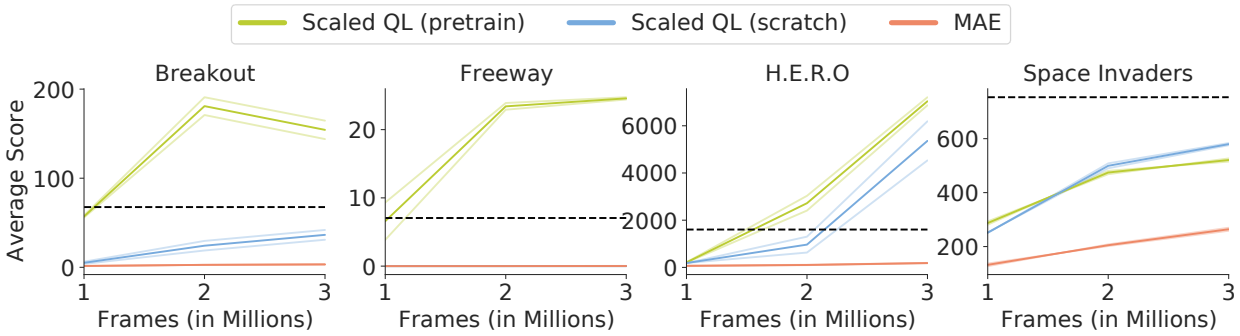


Figure 34: Learning curves for **online fine-tuning on unseen game variants**. The dotted horizontal line shows the performance of a single-game DQN agent trained for 50M frames (16x more data than our methods). See Figure 26 for visualization of the variants.

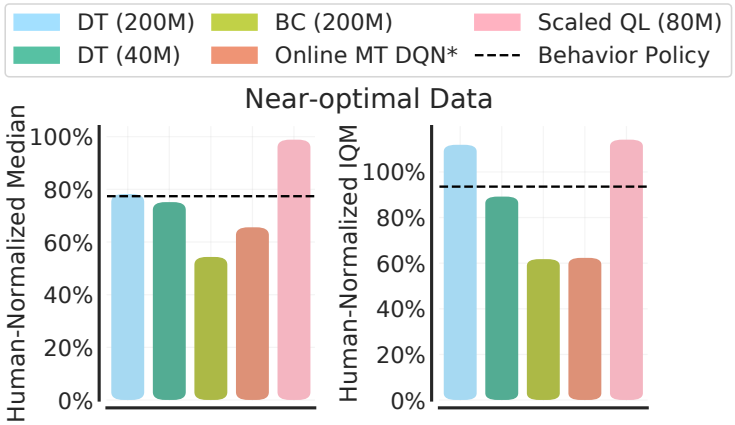


Figure 35: Offline scaled conservative Q-learning vs other prior methods with near-optimal data. Scaled QL outperforms the best DT model, attaining an IQM human-normalized score of **114.1%** and a median human-normalized score of **98.9%** compared to 111.8% and 78.2% for DT, respectively.

D.7.1 *Results for Scaling Discrete-BCQ*

To implement discrete BCQ, we followed the official implementation from Fujimoto et al. [98]. We first trained a model of the behavior policy, $\widehat{\pi}_\beta(\boldsymbol{a}|\boldsymbol{s})$, using an architecture identical to that of the Q-function, using negative log-likelihood. Then, following Fujimoto et al. [98], we updated the Bellman backup to only perform the maximization over actions that attain a high likelihood under the probabilities learned by the behavior policy, as shown below:

$$y(\boldsymbol{s}, \boldsymbol{a}) := r(\boldsymbol{s}, \boldsymbol{a}) + \gamma \max_{\boldsymbol{a}': \widehat{\pi}_\beta(\boldsymbol{a}'|\boldsymbol{s}') \geq \tau \cdot \max_{\boldsymbol{a}''} \widehat{\pi}_\beta(\boldsymbol{a}''|\boldsymbol{s}')} Q(\boldsymbol{s}', \boldsymbol{a}'),$$

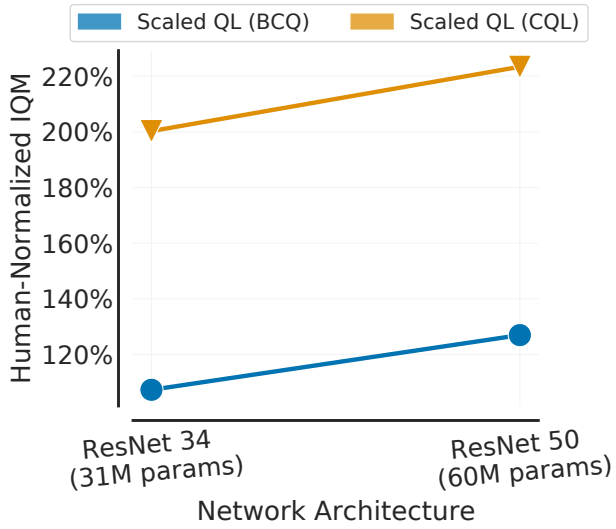


Figure 36: Performance of scaling CQL and BCQ in terms of IQM human-normalized score. We perform this comparison on the six-game setting for 100 epochs (note that these results are after 2x longer training than other ablations in Table 10). Observe that for discrete-BCQ the performance improves from ResNet 34 to ResNet 50, indicating that it does scale favorably as network capacity increases.

where τ is a hyperparameter. To tune the value of τ, we ran a preliminary initial sweep over $\tau = \{0.05, 0.1, 0.3\}$. When using C51 in our setup, we had to use a smaller CQL α of 0.05 (instead of 0.1 for the MSE setting from Kumar et al. [183]), possibly because a discrete representation of Q-values used by C51 is less prone to overestimation. Therefore, in the case of discrete-BCQ, we chose to perform an initial sweep over τ values that were smaller than or equal to (i.e., less conservative) the value of $\tau = 0.3$ used in Fujimoto et al. [98].

Since BCQ requires an additional policy network, it imposes a substantial memory overhead and as such, we performed a sweep for initial 20 iterations to pick the best τ. We found that in these initial experiments, $\tau = 0.05$ performed significantly worse, but $\tau = 0.1$ and $\tau = 0.3$ performed similarly. So, we utilized $\tau = 0.3$ for reporting these results.

We ran these scaling experiments with ResNet 34 and ResNet 50 in the six-game setting and report human-normalized IQM performance after 100 epochs = 6.25M gradient steps in Figure 36. We also present the results for CQL on the side for comparisons. Observe that we find favorable scaling trends for BCQ: average performance over all games increases as the network size increases, indicating that other offline RL algorithms such as BCQ can scale as we increase network capacity.

D.7.2 *Ablation for Backbone Architecture*

In this section, we present some results ablating the choice of the backbone architecture. For this ablation, we ablate the choice of the spatial embedding while keeping group normalization fixed in both cases. We perform this study for the 40-game setting. Observe that using the learned spatial embedding results in better performance, and improves in 27 out of 40 games compared to not using the learned embeddings.

Table 43: Ablations for the backbone architecture in the 40-game setting with ResNet 101. Observe that learned spatial embeddings leads to around 80% improvement in performance.

	Scaled QL without backbone	**Scaled QL w/ backbone**
Median human-normalized score	54.9%	**98.9**%
IQM human-normalized score	68.9%	**114.1**%
Num. games with better performance	13 / 40	**27 / 40**

Regarding the choice of group normalization vs batch normalization, note that we have been operating in a setting where the size of the batch per device / core is only 4. Particularly, we use Cloud TPU v3 accelerators with 64 / 128 cores, and bigger batch sizes than 4 do not fit in memory, especially for larger-capacity ResNets. This means that if we utilized batch normalization, we would be computing batch statistics over only 4 elements, which is known to be unstable even for standard computer vision tasks, for example, see Figure 1 in Wu and He [344].

D.7.3 *Results for Scaled QL Without Pessimism*

In Table 44, we present the results of running scaled Q-learning with no conservatism, i.e., by setting the value of α in CQL training to 0.0 in the six game setting. We utilize the entire DQN-replay dataset [6] for each of these six games that would be present in the full 40-game dataset, to preserve the per-game dataset diversity.

Observe that while utilizing no conservatism does still learn, the performance of scaled QL without conservatism is notably worse than standard scaled QL. Interestingly, on Asterix, the performance without pessimism is better than performance with pessimism, whereas, the use of pessimism in SpaceInvaders and Seaquest leads to at least 2x improvement in performance.

We also present some results without pessimism in the complete 40-game setting in Table 45. Unlike the smaller six game setting, we find a much larger difference between no pessimism (without CQL) and utilizing pessimism via CQL. In particular, we find that in 6 games, not using pessimism leads to slightly better performance, but this strategy hurts in all other games, giving rise to an agent that performs worse than random in many of these 34 games. This indicates that pessimism is especially deisrable as the diversity of tasks increases.

Table 44: Performance of scaled QL with and without conservatism in terms of IQM human-normalized score in the six-game setting for 100 epochs (2x longer training compared to other ablations in Table 10) performed with a ResNet 50. Observe that utilizing conservatism via CQL is beneficial. We also present per-game raw scores in this table. Observe that while in one games no pessimism with such data can outperform CQL, we do find that overall, conservatism performs better.

	Scaled QL without CQL	**Scaled QL w/ CQL**
Asterix	38000	35200
Breakout	322	410
Pong	12.6	19.8
Qbert	13800	15500
Seaquest	1378	3694
SpaceInvaders	1675	3819
IQM human-normalized score	188.3%	**223.4%**

Table 45: Scaled QL with and without conservatism in terms of IQM human-normalized score in the 40-game setting with ResNet 101. Observe that utilizing conservatism via CQL is still beneficial.

	Scaled QL without CQL	**Scaled QL w/ CQL**
Median human-normalized score	11.1%	98.9%
IQM human-normalized score	13.5%	114.1%
Num. games with better performance	6 / 40	**34 / 40**

D.8 Implementation details and hyper-parameters

In this section, we will describe some of the implementation details behind our method and will provide implementation details for our approach, including the details of the network architectures, the details of feature normalization and the details of our training and evaluation protocol.

D.8.1 *Network Architecture*

In our primary experiments, we consider variants of ResNet architectures for scaled Q-Learning. The vision backbone in these architectures mimic the corresponding ResNet architectures from He et al. [131], however, we utilize group normalization [344] (with a group size of 4) instead of batch normalization, and instead of applying global mean pooling to aggregate the outputs of the ResNet, we utilize learned spatial embeddings [188], that learn a matrix that point-wise multiplies the output feature map of the ResNet. The output volume is then flattened to be passed as input to the feed-forward part of the network. The feed-forward layer part of the network begins with a layer of size 2048, and

then applies layer norm on the network. After this we apply 3 feed-forward layers with hidden dimension 1024 and ReLU activations, to obtain the image representation

Then, we apply feature normalization to the representation, by applying a normalization layer which divides the representation of a given observation by its ℓ_2 norm. Note that we do pass gradients through this normalization term. Now, this representation is passed into different heads that are supposed to predict the Q-values. The total number of heads is equal to the number of games we train on. Each head consists of a linear layer that maps the 1024-dimensional normalized representation to a vector of K elements, where $K = |\mathcal{A}|$ (i.e., the size of the action space) for the standard real-valued parameterization of Q-values, and $K = |\mathcal{A}| \times 51$ for C51. The network does not apply any output activation in either case, and the Q-values are treated as logits for C51.

D.8.2 *Details of C51*

For the main results in the chapter, we utilize C51. The main hyperparameter in C51 is the size of the support set of Q-values. Unlike the paper from Bellemare et al. [24] which utilizes a support set of $[-10, 10]$, we utilize a support set of $[-20, 20]$ to allow for flexibility of CQL: Applying the CQL regularizer can underestimate or overestimate Q-values, and this additional flexibility aids such scenarios. Though, we still utilize only 51 atoms in our support set, and the average dataset Q-value in our training runs is generally always smaller, around $\sim 8 - 9$.

D.8.3 *Training and Evaluation Protocols and Hyperparameters*

We utilize the initial 20% (sub-optimal) and 100% (near-optimal) datasets from Agarwal et al. [6] for our experiments. These datasets are generated from runs of standard online DQN on stochastic dynamics Atari environments that utilize sticky actions, *i.e.* there is a 25% chance at every time step that the environment will execute the agents previous action again, instead of the new action commanded. The majority of the training details are identical to a typical run of offline RL on single-game Atari. We discuss the key differences below.

We trained our ResNet 101 network for $10M$ gradient steps with a batch size of 512. The agent hasn't converged yet, and the performance is still improving gradually. When training on multiple games, we utilize a stratified batch sampling scheme with a total batch size of 512. To obtain the batch at any given training iteration, we first sample 128 game indices from the set all games (40 games in our experiments) with replacement, and then sample 4 transitions from each game. This scheme does not necessarily produce an equal number of transitions from each game in a training batch, but it does make sure that all games are seen in expectation throughout training.

Table 46: Hyperparameters used by multi-game training. Here we report the key hyperparameters used by the multi-game training. The differences from standard single-game training are highlighted in red.

Hyperparameter	Setting (for both variations)
Eval Sticky actions	No
Grey-scaling	True
Observation down-sampling	(84, 84)
Frames stacked	4
Frame skip (Action repetitions)	4
Reward clipping	[-1, 1]
Terminal condition	Game Over
Max frames per episode	108K
Discount factor	0.99
Mini-batch size	512
Target network update period	every 2000 updates
Training environment steps per iteration	62.5k
Update period every	1 environment steps
Evaluation ϵ	0.001
Evaluation steps per iteration	125K
Learning rate	0.0002
n-step returns (n)	3
CQL regularizer weight α	0.1 for MSE, 0.05 for C51

Since we utilize a larger batch size, that is 16 times larger than the standard batch size of 32 on Atari, we scale up the learning rate from $5e-05$ to 0.0002, but keep the target network update period fixed to the same value of 1 target update per 2000 gradient steps as with single-task Atari. We also utilize n-step returns with $n = 3$ by default, with both our MSE and C51 runs.

Evaluation Protocol. Even though we train on Atari datasets with sticky actions, we evaluate on Atari environments that do not enable sticky actions following the protocol from Lee et al. [199]. This allows us to be comparable to this prior work in all of our comparisons, without needing to re-train their model, which would have been too computationally expensive. Following standard protocols on Atari, we evaluate a noised version of the policy with an epsilon-greedy scheme, with $\varepsilon_{\text{eval}} = 0.001$. Following the protocol in Castro et al. [38], we compute average return over 125K training steps.

D.8.4 *Fine-Tuning Protocol*

For offline fine-tuning we fine-tuned the parameters of the pre-trained policy on the new domain using a batch size of 32, and identical hyperparameters as those used during pre-training. We utilized $\alpha = 0.05$ for fine-tuning, but with the default learning rate

of $5e-05$ (since the batch size was the default 32). We attempted to use other CQL α values $\{0.07, 0.02, 0.1\}$ for fine-tuning but found that retaining the value of $\alpha = 0.05$ for pre-training worked the best. For reporting results, we reported the performance of the algorithm at the end of 300k gradient steps.

For online fine-tuning, we use the C51 algorithm [24], with n-step$= 3$ and all other hyperparameters from the C51 implementation in the Dopamine library [38]. We swept over two learning rates, $\{1e-05, 5e-05\}$ for all the methods and picked the best learning rate per-game for all the methods. For the MAE implementation, we used the Scenic library [59] with the typical configuration used for ImageNet pretraining, except using $84 \times 84 \times 4$ sized Atari observations, instead of images of size $224 \times 224 \times 3$. We train the MAE for 2 epochs on the entire multi-task offline Atari dataset and we observe that the reconstruction loss plateaus to a low value.

D.8.5 *Details of Multi-Task Impala DQN*

The "MT Impala DQN" comparison in Figures 20 & 23 is a multi-task implementation of online DQN, evaluated at 5x many gradient steps as the size of the sub-optimal dataset. This comparison is taken directly from Lee et al. [199]. To explain this baseline briefly, this baseline runs C51 in conjunction with n-step returns with $n = 4$, with an IMPALA architecture that uses three blocks with 64, 128, and 128 channels. This baseline was trained with a batch size of 128 and update period of 256.

D.9 RAW TRAINING SCORES FOR DIFFERENT METHODS

Table 47: Raw scores on 40 training Atari games in the sub-optimal multi-task Atari dataset (51% human-normalized IQM). Scaled QL uses the ResNet-101 architecture.

Game	DT (200M)	DT (40M)	Scaled QL (80M)	BC (80M)	MT Impala-DQN*	Human
Amidar	72.9	82.2	33.1	14.5	629.8	1719.5
Assault	392.9	124.7	1380.8	1060.0	1338.7	742.0
Asterix	1518.8	2256.2	9967.3	745.3	2949.1	8503.3
Atlantis	10525.0	13125.0	485200.0	2494.1	976030.4	29028.1
BankHeist	13.1	15.6	18.6	87.6	1069.6	753.1
BattleZone	3750.0	7687.5	8500.0	1550.0	26235.2	37187.5
BeamRider	1535.8	1397.5	5856.5	327.2	1524.8	16926.5
Boxing	71.4	74.2	95.2	95.4	68.3	12.1
Breakout	38.8	38.2	351.1	274.7	32.6	30.5
Carnival	993.8	791.2	199.3	792.7	2021.2	3800.0
Centipede	2645.4	3026.9	2711.4	2260.8	4848.0	12017.0
ChopperCommand	1006.2	1093.8	752.2	336.7	951.4	7387.8
CrazyClimber	85487.5	86050.0	122933.3	121394.4	146362.5	35829.4
DemonAttack	2269.7	1049.4	14229.8	765.8	446.8	1971.0
DoubleDunk	-14.5	-20.2	-12.4	-13.6	-156.2	-16.4
Enduro	336.5	266.2	2297.6	638.7	896.3	860.5
FishingDerby	15.9	16.8	13.7	-88.1	-152.3	-38.7
Freeway	16.2	20.5	24.4	0.1	30.6	29.6
Frostbite	1014.4	776.2	2324.5	234.8	2748.4	4334.7
Gopher	1137.5	1251.2	1041.0	231.5	3205.6	2412.5
Gravitar	237.5	193.8	260.3	248.8	492.5	3351.4
Hero	6741.2	6295.3	4011.9	7485.8	26568.8	30826.4
IceHockey	-8.8	-11.1	-3.7	-10.8	-10.4	0.9
Jamesbond	378.1	312.5	58.7	7.1	264.6	302.8
Kangaroo	1975.0	2687.5	5796.6	307.1	7997.1	3035.0
Krull	6913.8	4377.5	9333.7	9585.3	8221.4	2665.5
KungFuMaster	17575.0	14743.8	24320.0	15778.6	29383.1	22736.3
NameThisGame	4396.9	4502.5	6759.6	2756.8	6548.8	8049.0
Phoenix	3560.0	2813.8	12770.0	762.9	3932.5	7242.6
Pooyan	1053.8	1394.7	1264.5	718.7	4000.0	4000.0
Qbert	8371.9	5917.2	14877.9	5759.6	4226.5	13455.0
Riverraid	6191.9	4265.6	9602.7	6657.2	7306.6	17118.0
Robotank	14.9	12.8	17.4	5.7	9.2	11.9
Seaquest	781.9	512.5	1021.8	113.9	1415.2	42054.7
TimePilot	2512.5	2700.0	767.3	3841.1	-883.1	5229.2
UpNDown	5288.8	5456.2	35541.3	8395.2	8167.6	11693.2
VideoPinball	1277.4	1953.1	40.0	2650.3	85351.0	17667.9
WizardOfWor	237.5	881.2	107.0	495.3	975.9	4756.5
YarsRevenge	11867.4	10436.8	11482.4	17755.5	18889.5	54576.9
Zaxxon	287.5	337.5	1.4	0.0	-0.1	9173.3

Table 48: Raw scores on 40 training Atari games in the near-optimal multi-task Atari dataset. Scaled QL uses the ResNet 101 architecture.

Game	DT (200 M)	DT (40M)	BC (200M)	MT Impala-DQN*	Scaled QL (80M)	Human
Amidar	101.5	1703.8	101.0	629.8	21.0	1719.5
Assault	2385.9	1772.2	1872.1	1338.7	3809.6	742.0
Asterix	14706.3	4575.0	5162.5	2949.1	34278.9	8503.3
Atlantis	3105342.3	304931.2	4237.5	976030.4	881980.0	29028.1
BankHeist	5.0	40.0	63.1	1069.6	33.9	753.1
BattleZone	17687.5	17250.0	9250.0	26235.2	8812.5	37187.5
BeamRider	8560.5	3225.5	4948.4	1524.8	10301.0	16926.5
Boxing	95.1	92.1	90.9	68.3	99.5	12.1
Breakout	290.6	160.0	185.6	32.6	415.0	30.5
Carnival	2213.8	3786.9	2986.9	2021.2	926.1	3800.0
Centipede	2463.0	2867.5	2262.8	4848.0	3168.2	12017.0
ChopperCommand	4268.8	3337.5	1800.0	951.4	832.2	7387.8
CrazyClimber	126018.8	113425.0	123350.0	146362.5	140500.0	35829.4
DemonAttack	23768.4	3629.4	7870.6	446.8	56318.3	1971.0
DoubleDunk	-10.6	-12.5	-1.5	-156.2	-13.1	-16.4
Enduro	1092.6	770.8	793.2	896.3	2345.8	860.5
FishingDerby	11.8	19.2	5.6	-152.3	23.8	-38.7
Freeway	30.4	32.8	29.8	30.6	31.9	29.6
Frostbite	2435.6	934.4	782.5	2748.4	3566.4	4334.7
Gopher	9935.0	3827.5	3496.2	3205.6	3776.9	2412.5
Gravitar	59.4	75.0	12.5	492.5	262.3	3351.4
Hero	20408.8	19667.2	13850.0	26568.8	20470.6	30826.4
IceHockey	-10.1	-5.2	-8.3	-10.4	-1.5	0.9
Jamesbond	700.0	712.5	431.2	264.6	483.6	302.8
Kangaroo	12700.0	11581.2	12143.8	7997.1	2738.6	3035.0
Krull	8685.6	8295.6	8058.8	8221.4	10176.9	2665.5
KungFuMaster	15562.5	16387.5	4362.5	29383.1	25808.3	22736.3
NameThisGame	9056.9	7777.5	7241.9	6548.8	11647.0	8049.0
Phoenix	5295.6	4744.4	4326.9	3932.5	5264.0	7242.6
Pooyan	2859.1	1191.9	1677.2	4000.0	2020.1	4000.0
Qbert	13734.4	12534.4	11276.6	4226.5	15946.0	13455.0
Riverraid	14755.6	11330.6	9816.2	7306.6	18494.8	17118.0
Robotank	63.2	50.9	44.6	9.2	53.2	11.9
Seaquest	5173.8	3112.5	1175.6	1415.2	414.1	42054.7
TimePilot	2743.8	3487.5	1312.5	-883.1	4220.5	5229.2
UpNDown	16291.3	9306.9	10454.4	8167.6	55512.9	11693.2
VideoPinball	1007.7	9671.4	1140.8	85351.0	285.7	17667.9
WizardOfWor	187.5	687.5	443.8	975.9	301.6	4756.5
YarsRevenge	28897.9	25306.3	20738.9	18889.5	24393.9	54576.9
Zaxxon	275.0	4637.5	50.0	-0.1	2.1	9173.3

E

APPENDIX: OFFLINE RL WITH MULTI-TASK AND UNLABELED DATA

Conservative Data Sharing

E.1 PSEUDOCODE OF CDS

We present the summary of the pseudocode of CDS in Algorithm 6.

Algorithm 6 CDS: Conservative Data Sharing

Require: Multi-task offline dataset $\cup_{i=1}^{N} \mathcal{D}_i$.
1: Randomly initialize policy $\pi_\theta(\boldsymbol{a}|\boldsymbol{s}, i)$.
2: **for** $k = 1, 2, 3, \cdots,$ **do**
3: Initialize $\mathcal{D}^{\text{eff}} \leftarrow \{\}$
4: **for** $i = 1, \cdots, N$ **do**
5: $\mathcal{D}_i^{\text{eff}} = \mathcal{D}_i \cup \{(\boldsymbol{s}_j, \boldsymbol{a}_j, \boldsymbol{s}'_j, r_i) \in \mathcal{D}_{j \to i} : \Delta^{\pi}(\boldsymbol{s}, \boldsymbol{a}) \geq 0\}$ using Eq. 7.2.6 (CDS).
6: **end for**
7: Improve policy using samples from $\mathcal{D}^{\text{eff}}$ to obtain π_θ^{k+1}.
8: **end for**

E.2 ANALYSIS OF CDS

In this section, we will analyze the key idea behind our method CDS (Section 7.2.4) and show that the abstract version of our method (Equation 7.2.5) provides better policy improvement guarantees than naïve data sharing and that the practical version of our method (Equation 7.2.6) approximates Equation 7.2.5 resulting in an effective practical algorithm.

E.2.1 *Analysis of the Algorithm in Equation 7.2.5*

We begin with analyzing Equation 7.2.5, which is used to derive the practical variant of our method, CDS. We build on the analysis of safe-policy improvement guarantees of conventional offline RL algorithms [194, 181] and show that data sharing using CDS attains better guarantees in the worst case. To begin the analysis, we introduce some notation and prior results that we will directly compare to.

NOTATION AND PRIOR RESULTS. Let $\pi_\beta(\boldsymbol{a}|\boldsymbol{s})$ denote the behavior policy for task i (note that index i was dropped from $\pi_\beta(\boldsymbol{a}|\boldsymbol{s};i)$ for brevity). The dataset, $\mathcal{D}_i$ is generated from the marginal state-action distribution of π_β, i.e., $\mathcal{D} \sim d^{\pi_\beta}(\boldsymbol{s})\pi_\beta(\boldsymbol{a}|\boldsymbol{s})$. We define $d^{\pi}_{\mathcal{D}}$ as the state marginal distribution introduced by the dataset $\mathcal{D}$ under π. Let $D_{\text{CQL}}(p, q)$ denote the following distance between two distributions $p(\boldsymbol{x})$ and $q(\boldsymbol{x})$ with equal support $\mathcal{X}$:

$$D_{\text{CQL}}(p, q) := \sum_{\boldsymbol{x}\in\mathcal{X}} p(\boldsymbol{x}) \left(\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})} - 1\right).$$

Unless otherwise mentioned, we will drop the subscript "CQL" from D_{CQL} and use D and D_{CQL} interchangeably. Prior works [181] have shown that the optimal policy π^*_i that optimizes Equation 7.2.1 attains a high probability safe-policy improvement guarantee, i.e., $J(\pi^*_i) \geq J(\pi_\beta) - \zeta_i$, where ζ_i is:

$$\zeta_i = \mathcal{O}\left(\frac{1}{(1-\gamma)^2}\right) \mathbb{E}_{\boldsymbol{s}\sim d^{\pi^*_i}_{\mathcal{D}_i}} \left[\sqrt{\frac{D_{\text{CQL}}(\pi^*_i, \pi_\beta)(\boldsymbol{s}) + 1}{|\mathcal{D}_i(\boldsymbol{s})|}}\right] + \alpha D(\pi^*_i, \pi_\beta). \tag{E.2.1}$$

The first term in Equation E.2.1 corresponds to the decrease in performance due to sampling error and this term is high when the single-task optimal policy π^*_i visits rarely observed states in the dataset $\mathcal{D}_i$ and/or when the divergence from the behavior policy π_β is higher under the states visited by the single-task policy $\boldsymbol{s} \sim d^{\pi^*_i}_{\mathcal{D}_i}$.

Let $J_{\mathcal{D}}(\pi)$ denote the return of a policy π in the empirical MDP induced by the transitions in the dataset $\mathcal{D}$. Further, let us assume that optimizing Equation 7.2.5 gives us the following policies:

$$\pi^*(\boldsymbol{a}|\boldsymbol{s}), \pi^*_\beta(\boldsymbol{a}|\boldsymbol{s}) := \arg\max_{\pi,\pi_\beta\in\Pi_{\text{relabel}}} \underbrace{J_{\mathcal{D}^{\text{eff}}_i}(\pi) - \alpha D(\pi, \pi_\beta)}_{:= f(\pi,\pi_\beta;\mathcal{D}^{\text{eff}}_i)}, \tag{E.2.2}$$

where the optimized behavior policy π^*_β is constrained to lie in a set of all policies that can be obtained via relabeling, Π_{relabel}, and the dataset, $\mathcal{D}^{\text{eff}}_i$ is sampled according to the state-action marginal distribution of π^*_β, i.e., $\mathcal{D}^{\text{eff}}_i \sim d^{\pi^*_\beta}(\boldsymbol{s}, \boldsymbol{a})$. Additionally, for convenience, define, $f(\pi_1, \pi_2; \mathcal{D}) := J_{\mathcal{D}}(\pi_1) - \alpha D(\pi_1, \pi_2)$ for any two policies π_1 and π_2, and a given dataset $\mathcal{D}$.

We now show the following result for CDS:

Proposition E.2.1 (Proposition 7.2.1 restated). *Let $\pi^*(\boldsymbol{a}|\boldsymbol{s})$ be the policy obtained by optimizing Equation 7.2.5, and let $\pi_\beta(\boldsymbol{a}|\boldsymbol{s})$ be the behavior policy for $\mathcal{D}_i$. Then, w.h.p. $\geq 1-\delta$, π^* is a ζ-safe policy improvement over π_β, i.e., $J(\pi^*) \geq J(\pi_\beta) - \zeta$, where ζ is given by:*

$$\zeta = \mathcal{O}\left(\frac{1}{(1-\gamma)^2}\right) \mathbb{E}_{\boldsymbol{s}\sim d^{\pi^*}_{\mathcal{D}_i^{\text{eff}}}} \left[\sqrt{\frac{D_{CQL}(\pi^*, \pi^*_\beta)(\boldsymbol{s}) + 1}{|\mathcal{D}_i^{\text{eff}}(\boldsymbol{s})|}}\right] - \left[\alpha D(\pi^*, \pi^*_\beta) + \underbrace{J(\pi^*_\beta) - J(\pi_\beta)}_{(a)}\right],$$

where $\mathcal{D}_i^{\text{eff}} \sim d^{\pi^*_\beta}(\boldsymbol{s})$ and $\pi^*_\beta(\boldsymbol{a}|\boldsymbol{s})$ denotes the policy $\pi \in \Pi_{relabel}$ that maximizes Equation 7.2.5.

Proof. To prove this proposition, we shall quantify the lower-bound on the improvement in the policy performance due to Equation E.2.2 in the empirical MDP, and the potential drop in policy performance in the original MDP due to sampling error, and combine the terms to obtain our bound. First note that for any given policy π, and a dataset $\mathcal{D}_i^{\text{eff}}$ with effective behavior policy $\pi_\beta(\boldsymbol{a}|\boldsymbol{s})$, the following bound holds [181]:

$$J(\pi) \geq J_{\mathcal{D}_i^{\text{eff}}}(\pi) - \mathcal{O}\left(\frac{1}{(1-\gamma)^2}\right) \mathbb{E}_{\boldsymbol{s}\sim d^{\pi}_{\mathcal{D}_i^{\text{eff}}}} \left[\sqrt{\frac{D_{\text{CQL}}(\pi, \pi^*_\beta)(\boldsymbol{s}) + 1}{|\mathcal{D}_i^{\text{eff}}(\boldsymbol{s})|}}\right], \tag{E.2.3}$$

where the $\mathcal{O}(\cdot)$ notation hides constants depending upon the concentration properties of the MDP [194] and $1-\delta$, the probability with which the statement holds. Next, we provide guarantees on policy improvement in the empirical MDP. To see this, note that the following statements on $f(\pi_1, \pi_2; \mathcal{D})$ are true:

$$\forall \pi' \in \Pi_{\text{relabel}},\ \ f(\pi^*, \pi^*_\beta; \mathcal{D}_i^{\text{eff}}) \geq f(\pi', \pi', \mathcal{D}_i^{\text{eff}}) \tag{E.2.4}$$

$$\implies \forall \pi' \in \Pi_{\text{relabel}},\ \ J_{\mathcal{D}_i^{\text{eff}}}(\pi^*) - \alpha D(\pi^*, \pi^*_\beta) \geq J_{\mathcal{D}_i^{\text{eff}}}(\pi'). \tag{E.2.5}$$

And additionally, we obtain:

$$\forall \pi' \in \Pi_{\text{relabel}},\ \ f(\pi^*, \pi^*_\beta; \mathcal{D}_i^{\text{eff}}) \geq f(\pi^*, \pi'; \mathcal{D}_i^{\text{eff}}), \tag{E.2.6}$$

$$\implies \forall \pi' \in \Pi_{\text{relabel}},\ \ D(\pi^*, \pi^*_\beta) \leq D(\pi^*, \pi'). \tag{E.2.7}$$

Utilizing E.2.5, we obtain that:

$$J_{\mathcal{D}_i^{\text{eff}}}(\pi^*) - J_{\mathcal{D}_i^{\text{eff}}}(\pi_\beta) \geq \alpha D(\pi^*, \pi^*_\beta) + \left(J_{\mathcal{D}_i^{\text{eff}}}(\pi^*_\beta) - J_{\mathcal{D}_i^{\text{eff}}}(\pi_\beta)\right) \tag{E.2.8}$$

$$\approx \alpha D(\pi^*, \pi^*_\beta) + \left(J(\pi^*_\beta) - J(\pi_\beta)\right), \tag{E.2.9}$$

where $\approx$ ignores sampling error terms that do not depend on distributional shift measures like D_{CQL} because π^*_β and π_β are behavior policies which generated the complete and part

of the dataset, and hence these terms are dominated by and subsumed into the sampling error for π^*. Combining Equations E.2.3 (by setting $\pi = \pi^*$) and E.2.8, we obtain the following safe-policy improvement guarantee for π^*: $J(\pi^*) - J(\pi_\beta) \geq \zeta$, where ζ is given by:

$$\zeta = \mathcal{O}\left(\frac{1}{(1-\gamma)^2}\right) \mathbb{E}_{\boldsymbol{s}\sim d^{\pi^*}_{\mathcal{D}_i^{\text{eff}}}}\left[\sqrt{\frac{D_{\text{CQL}}(\pi^*, \pi^*_\beta)(\boldsymbol{s}) + 1}{|\mathcal{D}_i^{\text{eff}}(\boldsymbol{s})|}}\right] - \left[\alpha D(\pi^*, \pi^*_\beta) + \underbrace{J(\pi^*_\beta) - J(\pi_\beta)}_{\text{(a)}}\right],$$

which proves the desired result. □

Proposition E.2.1 indicates that when optimizing the behavior policy with Equation 7.2.5, we can improve upon the conventional safe-policy improvement guarantee (Equation E.2.1) with standard single-task offline RL: not only do we improve via $D_{\text{CQL}}(\pi^*, \pi^*_\beta)$, since, $D_{\text{CQL}}(\pi^*, \pi^*_\beta) \leq D_{\text{CQL}}(\pi^*, \pi_\beta)$, which reduces sampling error, but utilizing this policy π^*_β also allows us to improve on term (a), since Equation E.2.2 optimizes the behavior policy to be close to the learned policy π^* and maximizes the learned policy return $J_{\mathcal{D}_i^{\text{eff}}}(\pi^*)$ on the effective dataset, thus providing us with a high lower bound on $J(\pi^*_\beta)$. We formalize this insight as Lemma E.2.2 below:

Lemma E.2.2. *For sufficiently large α, $J_{\mathcal{D}_i^{\text{eff}}}(\pi^*_\beta) \geq J_{\mathcal{D}_i^{\text{eff}}}(\pi_\beta)$ and thus $(a) \geq 0$.*

Proof. To prove this, we note that using standard difference of returns of two policies, we get the following inequality: $J_{\mathcal{D}_i^{\text{eff}}}(\pi^*_\beta) \geq J_{\mathcal{D}_i^{\text{eff}}}(\pi^*) - C\frac{R_{\max}}{1-\gamma} D_{\text{TV}}(\pi^*, \pi^*_\beta)$. Moreover, from Equation E.2.5, we obtain that: $J_{\mathcal{D}_i^{\text{eff}}}(\pi^*) - \alpha D(\pi^*, \pi^*_\beta) \geq J_{\mathcal{D}_i^{\text{eff}}}(\pi_\beta)$. So, if α is chosen such that:

$$\frac{C R_{\max}}{1-\gamma} D_{\text{TV}}(\pi^*, \pi^*_\beta) \leq \alpha D(\pi^*, \pi^*_\beta), \tag{E.2.10}$$

we find that:

$$J_{\mathcal{D}_i^{\text{eff}}}(\pi^*_\beta) \geq J_{\mathcal{D}_i^{\text{eff}}}(\pi^*) - C\frac{R_{\max}}{1-\gamma} D_{\text{TV}}(\pi^*, \pi^*_\beta) \geq J_{\mathcal{D}_i^{\text{eff}}}(\pi^*) - \alpha D(\pi^*, \pi^*_\beta) \geq J_{\mathcal{D}_i^{\text{eff}}}(\pi_\beta),$$

implying that $(a) \geq 0$. For the edge cases when either $D_{\text{TV}}(\pi^*, \pi^*_\beta) = 0$ or $D_{\text{CQL}}(\pi^*, \pi^*_\beta) = 0$, we note that $\pi^*(\boldsymbol{a}|\boldsymbol{s}) = \pi^*_\beta(\boldsymbol{a}|\boldsymbol{s})$, which trivially implies that $J_{\mathcal{D}_i^{\text{eff}}}(\pi^*_\beta) = J_{\mathcal{D}_i^{\text{eff}}}(\pi^*) \geq J_{\mathcal{D}_i^{\text{eff}}}(\pi_\beta)$, because π^* improves over π_β on the dataset. Thus, term (a) is positive for large-enough α and the bound in Proposition E.2.1 gains from this term additionally. □

Finally, we show that the sampling error term is controlled when utilizing Equation 7.2.5. We will show in Lemma E.2.3 that the sampling error in Proposition E.2.1 is controlled to be not much bigger than the error just due to variance, since distributional shift is bounded with Equation 7.2.5.

Lemma E.2.3. *If π^* and π^*_β obtained from Equation 7.2.5 satisfy, $D_{CQL}(\pi^*, \pi^*_\beta) \leq \varepsilon \ll 1$, then:*

$$(\$) := \mathbb{E}_{\boldsymbol{s}\sim d^{\pi^*}_{\mathcal{D}_i^{\text{eff}}}}\left[\sqrt{\frac{D_{CQL}(\pi^*, \pi^*_\beta)(\boldsymbol{s})+1}{|\mathcal{D}_i^{\text{eff}}(\boldsymbol{s})|}}\right] \leq (1+\varepsilon)^{\frac{1}{2}} \underbrace{\mathbb{E}_{\boldsymbol{s}\sim d^{\pi^*}_{\mathcal{D}_i^{\text{eff}}}}\left[\sqrt{\frac{1}{|\mathcal{D}_i^{\text{eff}}(\boldsymbol{s})|}}\right]}_{:=\textit{sampling error w/o distribution shift}}. \tag{E.2.11}$$

Proof. This lemma can be proved via a simple application of the Cauchy-Schwarz inequality. We can partition the first term as a sum over dot products of two vectors such that:

$$\begin{aligned}
(\$) &= \sum_{\boldsymbol{s}} \sqrt{d^{\pi^*}_{\mathcal{D}_i^{\text{eff}}}(\boldsymbol{s})(D_{\text{CQL}}(\pi^*, \pi^*_\beta)(\boldsymbol{s})+1)}\sqrt{\frac{d^{\pi^*}_{\mathcal{D}_i^{\text{eff}}}(\boldsymbol{s})}{|\mathcal{D}_i^{\text{eff}}(\boldsymbol{s})|}} \\
&\leq \sqrt{\left(\sum_{\boldsymbol{s}} d^{\pi^*}_{\mathcal{D}_i^{\text{eff}}}(\boldsymbol{s})(D_{\text{CQL}}(\pi^*, \pi^*_\beta)(\boldsymbol{s})+1)\right)\cdot\left(\sum_{\boldsymbol{s}}\frac{d^{\pi^*}_{\mathcal{D}_i^{\text{eff}}}(\boldsymbol{s})}{|\mathcal{D}_i^{\text{eff}}(\boldsymbol{s})|}\right)} \\
&= \sqrt{\mathbb{E}_{\boldsymbol{s}\sim d^{\pi^*}_{\mathcal{D}_i^{\text{eff}}}}\left[D_{\text{CQL}}(\pi^*, \pi^*_\beta)(\boldsymbol{s})+1\right]\mathbb{E}_{\boldsymbol{s}\sim d^{\pi^*}_{\mathcal{D}_i^{\text{eff}}}}\left[\frac{1}{|\mathcal{D}_i^{\text{eff}}(\boldsymbol{s})|}\right]} \\
&\leq (1+\varepsilon)^{0.5}\mathbb{E}_{\boldsymbol{s}\sim d^{\pi^*}_{\mathcal{D}_i^{\text{eff}}}}\left[\sqrt{\frac{1}{|\mathcal{D}_i^{\text{eff}}(\boldsymbol{s})|}}\right],
\end{aligned}$$

where we note that $\mathbb{E}_{\boldsymbol{s}\sim d^{\pi^*}_{\mathcal{D}_i^{\text{eff}}}}\left[D_{\text{CQL}}(\pi^*, \pi^*_\beta)(\boldsymbol{s})\right] = D_{\text{CQL}}(\pi^*, \pi^*_\beta) \leq \varepsilon$ (based on the given information in the Lemma) and that $\sqrt{\sum_i w_i \frac{1}{x_i}} \leq \sum_i w_i \frac{1}{\sqrt{x_i}}$ for $x_i, w_i > 0$ and $\sum_i w_i = 1$, via Jensen's inequality for concave functions. □

To summarize, combining Lemmas E.2.2 and E.2.3 with Proposition E.2.1, we conclude that utilizing Equation 7.2.5 controls the increase in sampling error due to distributional shift, and provides improvement guarantees on the learned policy beyond the behavior policy of the original dataset. We also briefly now discuss the comparison between CDS and complete data sharing. Complete data sharing would try to reduce sampling error by increasing $|\mathcal{D}_i^{\text{eff}}(\boldsymbol{s})|$, but then it can also increase distributional shift, $D_{\text{CQL}}(\pi^*, \pi^*_\beta)$ as discussed in Section 7.2.3. On the other hand, CDS increases the dataset size while also controlling for distributional shift (as we discussed in the analysis above), making it enjoy the benefits of complete data sharing and avoiding its pitfalls, intuitively. On the other hand, no data sharing will just incur high sampling error due to limited dataset size.

E.2.2 *From Equation 7.2.5 to Practical CDS (Equation 7.2.6)*

The goal of our practical algorithm is to convert Equation 7.2.5 to a practical algorithm while retaining the policy improvement guarantees derived in Proposition E.2.1. Since our algorithm does not utilize any estimator for dataset counts $|\mathcal{D}_i^{\text{eff}}(\boldsymbol{s})|$, and since we operate in a continuous state-action space, our goal is to retain the guarantees of increased return of π_β^*, while also avoiding sampling error.

With this goal, we first need to relax the state-distribution in Equation 7.2.5: while both $J_{\mathcal{D}_i^{\text{eff}}}(\pi)$ and $D_{\text{CQL}}(\pi, \pi_\beta)$ are computed as expectations under the marginal state-distribution of policy $\pi(\boldsymbol{a}|\boldsymbol{s})$ on the MDP defined by the dataset $\mathcal{D}_i^{\text{eff}}$, for deriving a practical method we relax the state distribution to use the dataset state-distribution $d^{\pi_\beta^*}$ and rewrite the objective in accordance with most practical implementations of actor-critic algorithms [58, 2, 126, 97, 211] below:

$$\text{(Practical Eq. 7.2.5)} \quad \max_{\pi} \max_{\pi_\beta \in \Pi_{\text{relabel}}} \mathbb{E}_{\boldsymbol{s} \sim \mathcal{D}_i^{\text{eff}}}[\mathbb{E}_{\boldsymbol{a} \sim \pi(\boldsymbol{a}|\boldsymbol{s})}[Q(\boldsymbol{s}, \boldsymbol{a})] - \alpha D(\pi(\cdot|\boldsymbol{s}), \pi_\beta(\cdot|\boldsymbol{s}))] \tag{E.2.12}$$

This practical approximation in Equation E.2.12 is even more justified with conservative RL algorithms when a large α is used, since a larger α implies a smaller value for $D(\pi^*, \pi_\beta^*)$ found by Equation 7.2.5, which in turn means that state-distributions $d^{\pi_\beta^*}$ and d^{π^*} are close to each other [290]. Thus, our policy improvement objective optimizes the policies π and π_β by maximizing the conservative Q-function: $\hat{Q}^\pi(\boldsymbol{s}, \boldsymbol{a}) = Q(\boldsymbol{s}, \boldsymbol{a}) - \alpha \left(\frac{\pi(\boldsymbol{a}|\boldsymbol{s})}{\pi_\beta(\boldsymbol{a}|\boldsymbol{s})} - 1 \right)$, that appears inside the expectation in Equation E.2.12. While optimizing the policy π with respect to this conservative Q-function $\hat{Q}^\pi(\boldsymbol{s}, \boldsymbol{a})$ is equivalent to a standard policy improvement update utilized by most actor-critic methods [97, 126, 181], we can optimize $\hat{Q}^\pi(\boldsymbol{s}, \boldsymbol{a})$ with respect to $\pi_\beta \in \Pi_{\text{relabel}}$ by relabeling only those transitions $(\boldsymbol{s}, \boldsymbol{a}, r_i', \boldsymbol{s}') \in \mathcal{D}_{j \to i}$ that increase the expected conservative Q-value $\mathbb{E}_{\boldsymbol{s} \sim \mathcal{D}_i^{\text{eff}}} \left[\mathbb{E}_{\boldsymbol{a} \sim \pi_\beta(\cdot|\boldsymbol{s})} \left[\hat{Q}^\pi(\boldsymbol{s}, \boldsymbol{a}) \right] \right]$. Note that we relaxed the expectation $\boldsymbol{a} \sim \pi(\boldsymbol{a}|\boldsymbol{s})$ to $\boldsymbol{a} \sim \pi_\beta(\boldsymbol{a}|\boldsymbol{s})$ in this expectation, which can be done upto a lower-bound of the objective in Equation E.2.12 for a large α, since the resulting policies π and π_β are close to each other.

The last step in our practical algorithm is to modify the solution of Equation E.2.12 to still retain the benefits of reduced sampling error as discussed in Proposition E.2.1. To do so, we want to relabel as many points as possible, thus increasing $|\mathcal{D}_i^{\text{eff}}(\boldsymbol{s})|$, which leads to reduced sampling error. Since quantifying $|\mathcal{D}_i^{\text{eff}}(\boldsymbol{s})|$ in continuous state-action spaces will require additional machinery such as density-models, we avoid these for the sake of simplicity, and instead choose to relabel every datapoint $(\boldsymbol{s}, \boldsymbol{a}) \in \mathcal{D}_{j \to i}$ that satisfies $Q^\pi(\boldsymbol{s}, \boldsymbol{a}; i) \geq \mathbb{E}_{\boldsymbol{s}, \boldsymbol{a} \sim \mathcal{D}_i}[\hat{Q}^\pi(\boldsymbol{s}, \boldsymbol{a}; i)] \geq 0$ to task i. These datapoints definitely increase the conservative Q-value and hence increase the objective in Equation E.2.12 (though do not

globally maximize it), while also enjoying properties of reduced sampling error (Proposition E.2.1). This discussion motivates our practical algorithm in Equation 7.2.6.

E.3 EXPERIMENTAL DETAILS

In this section, we provide the training details of CDS in Appendix E.3.1 and also include the details on the environment and datasets that we use for the evaluation in Appendix E.3.2. We also compare CDS to an offline RL with pretrained representations from multi-task datasets method [354].

E.3.1 *Training details*

The pseudocode of CDS is summarized in Algorithm 6 in Appendix E.1. The complete variant of CDS can be directly implemented using the rule in Equation 7.2.6 with conservative Q-value estimates obtained via any offline RL method that constrains the learned policy to the behavior policy. For implementing CDS (basic), we reparameterize the divergence D in Equation 7.2.4 to use the learned conservative Q-values. This is especially useful for our implementation since we utilize CQL as the base offline RL method, and hence we do not have access to an explicit divergence. In this case, $\Delta^{\pi}(\boldsymbol{s}, \boldsymbol{a})$ can be redefined as, $\Delta^{\pi}(\boldsymbol{s}, \boldsymbol{a}) :=$

$$\mathbb{E}_{\boldsymbol{s}' \sim \mathcal{D}^i}\left[\mathbb{E}_{\boldsymbol{a}' \sim \pi}[\hat{Q}(\boldsymbol{s}', \boldsymbol{a}', i)] - \mathbb{E}_{\boldsymbol{a}'' \sim \mathcal{D}_i}[\hat{Q}(\boldsymbol{s}', \boldsymbol{a}'', i)]\right] - \left(\mathbb{E}_{\boldsymbol{a}' \sim \pi}[\hat{Q}(\boldsymbol{s}, \boldsymbol{a}', i)] - Q(\boldsymbol{s}, \boldsymbol{a}, i)\right), \tag{E.3.1}$$

Equation E.3.1 can be viewed as the difference between the CQL [181] regularization term on a given $(\boldsymbol{s}, \boldsymbol{a})$ and the original dataset for task i, $\mathcal{D}_i$. This CQL regularization term is equal to the divergence between the learned policy $\pi(\cdot|\boldsymbol{s})$ and the behavior policy $\pi_\beta(\cdot|\boldsymbol{s})$, therefore Equation E.3.1 practically computes Equation 7.2.4.

Note that both variants of CDS train a policy, $\pi(\boldsymbol{a}|\boldsymbol{s}; i)$, either conditioned on the task i (i.e., with weight sharing) or a separate $\pi(\boldsymbol{a}|\boldsymbol{s})$ policy for each task with no weight sharing, using the resulting relabeled dataset, $\mathcal{D}_i^{\text{eff}}$. Next, we discuss the training details of the complete version of CDS.

Our practical implementation of CDS optimizes the following objectives for training the critic and the policy:

$$\begin{aligned}\hat{Q}^{k+1} \leftarrow \arg\min_{\hat{Q}} \mathbb{E}_{i \sim [N]} \Big[\beta \Big(\mathbb{E}_{j \sim [N]} \Big[\mathbb{E}_{\boldsymbol{s} \sim \mathcal{D}_j, \boldsymbol{a} \sim \mu(\cdot|\boldsymbol{s}, i)} \left[w_{\text{CDS}}(\boldsymbol{s}, \boldsymbol{a}; j \to i) \hat{Q}(\boldsymbol{s}, \boldsymbol{a}, i)\right] \\ - \mathbb{E}_{\boldsymbol{s}, \boldsymbol{a} \sim \mathcal{D}_j}\left[w_{\text{CDS}}(\boldsymbol{s}, \boldsymbol{a}; j \to i) \hat{Q}(\boldsymbol{s}, \boldsymbol{a}, i)\right]\Big]\Big) \\ + \frac{1}{2} \mathbb{E}_{j \sim [N], (\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}') \sim \mathcal{D}_j}\left[w_{\text{CDS}}(\boldsymbol{s}, \boldsymbol{a}; j \to i) \left(\hat{Q}(\boldsymbol{s}, \boldsymbol{a}, i) - \widehat{\mathcal{B}}^{\pi} \hat{Q}^k(\boldsymbol{s}, \boldsymbol{a}, i)\right)^2\right]\Big],\end{aligned}$$

and $$\pi \leftarrow \arg\max_{\pi'} \mathbb{E}_{i \sim [N]}\left[\mathbb{E}_{j \sim [N], \boldsymbol{s} \sim \mathcal{D}_j, \boldsymbol{a} \sim \pi'(\cdot|\boldsymbol{s}, i)}\left[w_{\text{CDS}}(\boldsymbol{s}, \boldsymbol{a}; j \to i) \hat{Q}^{\pi}(\boldsymbol{s}, \boldsymbol{a}, i)\right]\right],$$

where β is the coefficient of the CQL penalty on distribution shift, μ is a wide sampling distribution as in CQL and $\widehat{\mathcal{B}}$ is the sample-based Bellman operator.

To compute the relabeling weight $w_{\text{CDS}}(\boldsymbol{s}, \boldsymbol{a}; j \to i) := \sigma\left(\frac{\Delta(\boldsymbol{s},\boldsymbol{a};j\to i)}{\tau}\right)$, we need to pick the value of the temperature term τ. Instead of tuning τ manually, we follow the the adaptive temperature scaling scheme from [180]. Specifically, we compute an exponential running average of $\Delta(\boldsymbol{s}, \boldsymbol{a}; j \to i)$ with decay 0.995 for each task and use it as τ. We additionally clip the adaptive temperature term with a minimum and maximum threshold, which we tune manually. For multi-task halfcheetah, walker2d and ant, we clip the adaptive temperature such that it lies within $[10, \infty]$, $[5, \infty]$ and $[10, 25]$ respectively. For the multi-task Meta-Wold experiment, we use $[1, 50]$ for the clipping. For multi-task Antmaze, we used a range of $[10, \infty]$ for all the domains. We do not clip the temperature term on vision-based domains.

For state-based experiments, we use a stratified batch with 128 transitions for each task for the critic and policy learning. For each task i, we sample 64 transitions from $\mathcal{D}_i$ and another 64 transitions from $\cup_{j\neq i}\mathcal{D}_{j\to i}$, i.e. the relabeled datasets of all the other tasks. When computing $\Delta(\boldsymbol{s}, \boldsymbol{a}; j \to i)$, we only apply the weight to relabeled data on multi-task Meta-World environments and multi-task vision-based robotic manipulation tasks while also applying the weight to the original data drawn from $\mathcal{D}_i$ with 50% chance for each task $i \in [N]$ in the remaining domains.

We use CQL [181] as the base offline RL algorithm. On state-based experiments, we mostly follow the hyperparameters provided in prior work [181]. One exception is that on the multi-task ant domain, we set $\beta = 5.0$ and on the other two locomotion environments and the multi-task Meta-World domain, we use $\beta = 1.0$. On multi-task AntMaze, we use the Lagrange version of CQL, where the multiplier β is automatically tuned against a pre-specific constraint value on the CQL loss equal to $\tau = 5.0$. We use a policy learning rate $1e-4$ and a critic learning rate $3e-4$ as in [181]. On the vision-based environment, instead of using the direct CQL algorithm, we follow [41] and sample unseen actions according to the soft-max distritbution of the Q-values and set its Q target value to 0. This algorithm can be viewed the version of CQL with $\beta = 1.0$ in Eq.1 in [181], i.e. removing the term of negative expected Q-values on the dataset. We follow the other hyperparameters from prior work [161, 41, 162].

For the choice architectures, in the domains with low-dimensional state inputs, we use 3-layer feedforward neural networks with 256 hidden units for both the Q-networks and the policy. We append a one-hot task vector to the state of each environment. For the vision-based experiment, our Q-network architecture follows from multi-headed convolutional networks used in MT-Opt [162]. For the observation input, we use images with dimension $472 \times 472 \times 3$ along with additional state features $(g_{\text{status}}, g_{\text{height}})$ as well as the one-hot task vector as in [162]. For the action input, we use Cartesian space control of the end-effector of the robot in 4D space (3D position and azimuth angle) along

with two discrete actions for opening/closing the gripper and terminating the episode respectively. More details can be found in [161, 162].

E.3.2 *Environment and dataset details*

In this subsection, we discuss the details of how we set up the multi-task environment and how we collect the offline datasets. We want to acknowledge that all datasets with state inputs use the MIT License.

Multi-task locomotion domains. We construct the environment by changing the reward function in [32]. On the halfcheetah environment, we follow [365] and set the reward functions of task run forward, run backward and jump as $r(s,a) = \max\{v_x, 3\} - 0.1 * \|a\|_2^2$, $r(s,a) = -\max\{v_x, 3\} - 0.1 * \|a\|_2^2$ and $r(s,a) = -0.1 * \|a\|_2^2 + 15 * (z - \text{init z})$ respectively where v_x denotes the velocity along the x-axis and z denotes the z-position of the half-cheetah and init z denotes the initial z-position. Similarly, on walker2d, the reward functions of the three tasks are $r(s,a) = v_x - 0.001 * \|a\|_2^2$, $r(s,a) = -v_x - 0.001 * \|a\|_2^2$ and $r(s,a) = -\|v_x\| - 0.001 * \|a\|_2^2 + 10 * (z - \text{init z})$ respectively. Finally, on ant, the reward functions of the three tasks are $r(s,a) = v_x - 0.5 * \|a\|_2^2 - 0.005 *$ contact-cost, $r(s,a) = -v_x - 0.5 * \|a\|_2^2 - 0.005 *$ contact-cost and $r(s,a) = -\|v_x\| - 0.5 * \|a\|_2^2 - 0.005 *$ contact-cost $+ 10 * (z - \text{init z})$.

On each of the multi-task locomotion environment, we train each task with SAC [126] for 500 epochs. For medium-replay datasets, we take the whole replay buffer after the online SAC is trained for 100 epochs. For medium datasets, we take the online single-task SAC policy after 100 epochs and collect 500 trajectories with the medium-level policy. For expert datasets, we take the final online SAC policy and collect 5 trajectories with it for walker2d and halfcheetah and 20 trajectories for ant.

Meta-World domains. We take the door open, door close, drawer open and drawer close environments from the open-sourced Meta-World [364] repo[1]. We put both the door and the drawer on the same scene to make sure the state space of all four tasks are shared. For offline training, we use sparse rewards for each task by replacing the dense reward defined in Meta-World with the success condition defined in the public repo. Therefore, each task gets a reward of 1 if the task is fully completed and 0 otherwise.

For generating the offline datasets, we train each task with online SAC using the dense reward defined in Meta-World for 500 epochs. For medium-replay datasets, we take the whole replay buffer of the online SAC until 150 epochs. For the expert datasets, we run the final online SAC policy to collect 10 trajectories.

AntMaze domains. We take the antmaze-medium-play and antmaze-large-play datasets from D4RL [92] and convert the datasets into multi-task datasets in two ways. In the undirected version of these tasks, we split the dataset randomly into equal sized partitions,

[1] The Meta-World environment can be found at the public repo https://github.com/rlworkgroup/metaworld

and then assign each partition to a particular randomly chosen task. Thus, the task data observed in the data for each task is largely unsuccessful for the particular task it is assigned to and effective data sharing is essential for obtaining good performance. The second setting is the directed data setting where a trajectory in the dataset is marked to belong to the task corresponding to the actual end goal of the trajectory. A sparse reward equal to +1 is provided to an agent when the current state reaches within a 0.5 radius of the task goal as was used default by Fu et al. [92].

Vision-based robotic manipulation domains. Following MT-Opt [162], we use sparse rewards for each task, i.e. reward 1 for success episodes and 0 otherwise. We define successes using the success detectors defined in [162]. To collect data for vision-based experiments, we train a policy for each task individually by running QT-Opt [161] with default hyperparameters until the task reaches 40% success rate for picking skills and 80% success rate for placing skills. We take the whole replay buffer of each task and combine all of such replay buffers to form the multi-task offline dataset with total 100K episodes where each episode has 25 transitions.

E.4 VISUALIZATIONS, COMPARISONS AND ADDITIONAL EXPERIMENTS

In this section, we perform diagnostic and ablation experiments to: **(1)** understand the efficacy of CDS when applied with other base offline RL algorithms, such as BRAC [343], **(2)** visualize the weights learned by CDS to understand if the weighting scheme induced by CDS corresponds to what we would intuitively expect on different tasks, and **(3)** compare CDS to a prior approach that performs representation learning from offline multi-task datasets and then runs vanilla multi-task RL algorithm on top of the learned representations. We discuss these experiments next.

E.4.1 *Applying CDS with BRAC [343], A Policy-Constraint Offline RL Algorithm*

We implemented CDS on top of BRAC which is different from CQL that penalizes Q-functions. BRAC computes the divergence $D(\pi, \pi_\beta)$ in Equation 7.2.1 explicitly and penalizes the reward function $r(\boldsymbol{s}, \boldsymbol{a})$ with this value in the Bellman backups. To apply CDS to BRAC, we need to compute a conservative estimate of the Q-value as discussed in Section 7.2.4.2. While the Q-function from CQL directly provides us with this conservative estimate, BRAC does not directly learn a conservative Q-function estimator. Therefore, for BRAC, we compute this conservative estimate by explicitly subtracting KL divergence between the learned policy $\pi(a|s)$ and the behavior policy π^β on state-action tuples (s, a) from the learned Q-function's prediction. Formally, this means that we utilize $\hat{Q}(s, a) := Q(s, a) - \alpha \mathrm{D}_{\mathrm{KL}}(\pi(a|s), \pi^\beta(a|s))$ as our conservative Q-value estimate for BRAC. Given these conservative Q-value estimate, CDS weights can be computed directly using Equation 7.2.6.

Environment	Tasks / Dataset type	BRAC + CDS (ours)	BRAC + No Sharing	BRAC + Sharing All
Meta-World [364]	door open / expert	**44.0%**±3.0%	35.0%±25.9%	38.0%±2.2%
	door close / medium-replay	**32.5%** ± 5.0%	5.0% ± 8.6%	8.6% ± 3.4%
	drawer open / medium-replay	**28.5%**±3.5%	21.8%±5.6%	0.0% ± 0.0%
	drawer close / expert	100%±0.0%	100.0%±0.0%	99.0%±0.7%
	average	**52.5%**±7.4%	22.5%±13.3%	40.0%±5.0%

Table 49: Applying CDS on top of BRAC. Note that CDS + BRAC imrpoves over both BRAC + Sharing All and BRAC + No sharing, indicating that CDS is effective over other offline RL algorithms such as BRAC as well. The ± values indicate the value of the standard deviation of runs, and results are averaged over three seeds.

We evaluated BRAC + CDS on the Meta-World tasks and compared it to BRAC + Sharing All and BRAC + No Sharing. We present the results in Table 49. We use ± to denote the 95%-confidence interval. As observed below, BRAC + CDS significantly outperforms BRAC with Sharing All and BRAC with No sharing. This indicates that CDS is effective on top of BRAC.

E.4.2 *Analyzing CDS weights for Different Scenarios*

Next, to understand if the weights assigned by CDS align with our expectation for which transitions should be shared between tasks, we perform diagnostic analysis studies on the weights learned by CDS on the Meta-World and Antmaze domains.

On the Meta-World environment, we would expect that for a given target task, say Drawer Close, transitions from a task that involves a different object (door) and a different skill (open) would not be as useful for learning. To understand if CDS weights reflect this expectation, we compare the average CDS weights on transitions from all the other tasks to two target tasks, Door Open and Drawer Close, respectively and present the results in Table 50. We sort the CDS weights in the descending order. As shown, indeed CDS assigns higher weights to more related tasks and thus shares data from those tasks. In particular, the CDS weights for relabeling data from the task that handles the same object as the target task are much higher than the weights for tasks that consider a different object.

For example, when relabeling to the target task Door Open, datapoints from task Door Close are assigned with much higher weights than those from either task Drawer Open or task Drawer Close. This suggests that CDS filters the irrelevant transitions for learning a given task.

On the AntMaze-large environment, with undirected data, we visualize the CDS weight for the various tasks (goals) in the form of a heatmap and present the results in Figure 37. To generate this plot, we sample a set of state-action pairs from the entire dataset for all tasks, and then plot the weights assigned by CDS as the color of the point marker at the (x, y) locations of these state-action pairs in the maze. Each plot computes the CDS weight

Relabeling Direction	CDS weight
door close → door open	0.46
drawer open → door open	0.10
drawer close → door open	0.02
drawer open → drawer close	0.35
door open → drawer close	0.26
door close → drawer close	0.22

Table 50: On the Meta-World domain, we visualize the CDS weights of data relabeled from other tasks to the two target tasks door open and drawer close shown in the second row and third row respectively. We sort the CDS weights for relabeled tasks to a particular target task in the descending order. As shown in the table, CDS upweights tasks that are more related to the target task, e.g. manipulating the same object.

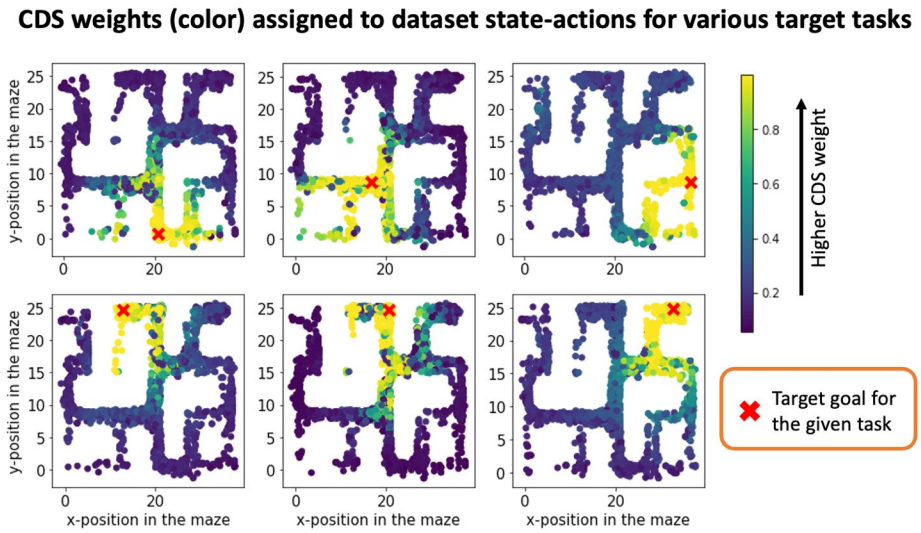


Figure 37: A visualization of the weights assigned by CDS to various transitions in the antmaze dataset for six target goals (indicated by $\times$ clustered by their spatial location. Note that CDS up-weights transitions spatially close to the target goal (indicated in the brighter yellow color), matching our expectation.

corresponding to the target task (goal) indicated by the red $\times$ in the plot. As can be seen in Figure 37, CDS assigns higher weights to transitions from nearby goals as compared to transitions from farther away goals. This matches our expectation: transitions from nearby (x, y) locations are likely to be the most useful in learning a particular target task and CDS chooses to share these transitions to the target task.

E.4.3 *Comparison of CDS with Other Alternatives to Data Sharing: Utilizing Multi-Task Datasets for Learning Pre-Trained Representations*

Finally, we aim to empirically verify how other alternatives to data sharing perform on multi-task offline RL problems. One simple approach to utilize data from other tasks is to use this data to learn low-dimensional representations that capture meaningful

Environment	Tasks / Dataset type	CDS (ours)	No Sharing	Offline Petraining [354]
Meta-World [364]	door open / expert	**58.4%**±9.3%	14.5%±12.7%	48.0%±40.0%
	door close / medium-replay	**65.3%**±27.7%	4.0%±6.1%	9.5%±8.4%
	drawer open / medium-replay	**57.9%**±16.2%	16.0%±17.5%	1.3% ± 1.4%
	drawer close / expert	98.8%±0.7%	99.0%±0.7%	96.0%±0.9%
	average	**70.1%**±8.1%	33.4%±8.3%	38.7%±11.1%

Table 51: Comparison between CDS and Offline Pretraining [354] that pretrains the representation from the multi-task offline data and then runs multi-task offline RL on top of the learned representation on the Meta-World domain. Numbers are averaged across 6 seeds, ± the 95%-confidence interval. CDS significantly outperforms Offline Pretraining.

information about the environment initially in a pre-training phase and then utilize these representations for improved multi-task RL without any specialized data sharing schemes. To assess the efficacy of this alternate approach of using multi-task offline data, in Table 51, we performed an experiment on the Meta-World domain that first utilizes the data from all the tasks to learn a shared representation using the best method, ACL [354] and then runs standard offline multi-task RL on top of this representation. We denote the method as **Offline Pretraining**. We include the average task success rates of all tasks in the table below. While the representation learning approach improves over standard multi-task RL without representation learning (**No Sharing**) consistent with the findings in [354], we still find that CDS with no representation learning outperforms this representation learning approach by a large margin on multi-task performance, which suggests that conservative data sharing is more important than pure pretrained representation from multi-task datasets in the offline multi-task setting. We finally remark that in principle, we could also utilize representation learning approaches in conjunction with data sharing strategies and systematically characterizing this class of hybrid approaches is a topic of future work.

E.5 MISSING PROOFS

In this section, we will theoretically analyze UDS and other reweighting schemes to better understand when these approaches can perform well. We will first discuss our notation, then present our theoretical results, then discuss the intuitive explanations of these results, and finally, provide proofs of the theoretical results.

E.5.1 *Performance Guarantee for UDS*

We first restate Proposition 7.3.1 below for convenience, and we then provide a proof of the result.

Theorem E.5.1 (Policy improvement guarantee for UDS; restated.**).** *Let π^*_{UDS} denote the policy learned by UDS, and let $\pi^{\text{eff}}_\beta(\boldsymbol{a}|\boldsymbol{s})$ denote the behavior policy for the combined dataset $\mathcal{D}^{\text{eff}}$. Then with high probability $\geq 1-\delta$, π^*_{UDS} is a safe policy improvement over π^{eff}_β, i.e.,*

$$J(\pi^*_{UDS}) \geq J(\pi^{\text{eff}}_\beta) - \zeta_{err} + \underbrace{\frac{\alpha}{1-\gamma} D(\pi^*_{UDS}, \pi^{\text{eff}}_\beta)}_{\text{(c): policy improvement}},$$

$$\text{where: } \zeta_{err} = \underbrace{\frac{\sum_{\boldsymbol{s},\boldsymbol{a}} \left(\widehat{d}^{\pi_{\beta^{\text{eff}}}}(\boldsymbol{s},\boldsymbol{a}) - \widehat{d}^{\pi^*_{UDS}}(\boldsymbol{s},\boldsymbol{a}) \right) \cdot (1 - f(\boldsymbol{s},\boldsymbol{a})) \cdot r(\boldsymbol{s},\boldsymbol{a})}{1-\gamma}}_{\text{(a): reward bias}}$$

$$+ \underbrace{\mathcal{O}\left(\frac{\gamma}{(1-\gamma)^2}\right) \left[\sqrt{\frac{D_{CQL}(\pi^*_{UDS}, \pi^{\text{eff}}_\beta)(\boldsymbol{s})}{|\mathcal{D}^{\text{eff}}(\boldsymbol{s})|}} \right]}_{\text{(b): sampling error}},$$

where we use the notation $f(\boldsymbol{s},\boldsymbol{a}) := \frac{|\mathcal{D}_{\text{L}}(\boldsymbol{s},\boldsymbol{a})|}{|\mathcal{D}^{\text{eff}}(\boldsymbol{s},\boldsymbol{a})|}$.

Proof. We start with the loss decomposition of the improvement of the learned policy relative to the behavior policy with the affine transformation g:

$$J(\pi) - J(\pi_\beta) := \underbrace{J(\pi) - \widehat{J}(\pi)}_{(i)} + \underbrace{\widehat{J}(\pi) - \widehat{J}(\pi_\beta)}_{(ii)} + \underbrace{\widehat{J}(\pi_\beta) - J(\pi_\beta)}_{(iii)}.$$

Now we will discuss how to bound each of the terms: terms (i) and (ii) correspond to the divergence between the empirical policy return and the actual return. While

usually, this difference depends on the sampling error and distributional shift, in our case, it additionally depends on the reward bias induced on the unlabeled data and the transformation g. We first discuss the terms that contribute to this reward bias.

BOUNDING THE REWARD BIAS. Denote the effective reward of a particular transition $(s, a, r, s') \in \mathcal{D}^{\text{eff}}$, as $\widehat{r}^{\text{eff}}$, which considers contributions from both the reward $\widehat{r}(s, a)$ observed in dataset $\mathcal{D}_{\text{L}}$, and the contribution of 0 reward from the relabeled dataset:

$$\widehat{r}^{\text{eff}}(s, a) = \frac{|\mathcal{D}(s, a)| \cdot \widehat{r}(s, a) + |\mathcal{D}^{\text{eff}}(s, a) \setminus \mathcal{D}(s, a)| \cdot 0}{|\mathcal{D}^{\text{eff}}(s, a)|} \tag{E.5.1}$$

Define $f(s, a) := \frac{|\mathcal{D}(s,a)|}{|\mathcal{D}^{\text{eff}}(s,a)|}$ for notation compactness. Equation E.5.1 can then be used to derive the following difference against the true rewards:

$$\widehat{r}^{\text{eff}}(s, a) - r(s, a) = f(s, a)\left(\widehat{r}(s, a) - r(s, a)\right) + (1 - f(s, a)) \cdot (0 - r(s, a)) \tag{E.5.2}$$

$$\leq f(s, a) \cdot \frac{C_{r,\delta}}{\sqrt{|\mathcal{D}(s, a)|}} - (1 - f(s, a)) \cdot r(s, a), \tag{E.5.3}$$

where the last step follows from the fact that the ground-truth reward $r(s, a) \in [0, 1]$. Now, we lower bound the reward bias as follows:

$$\widehat{r}^{\text{eff}}(s, a) + - r(s, a) = f(s, a) \cdot (\widehat{r}(s, a) - r(s, a)) + (1 - f(s, a)) \cdot (-r(s, a)) \tag{E.5.4}$$

$$\geq -f(s, a) \cdot \frac{C_{r,\delta}}{\sqrt{|\mathcal{D}(s, a)|}} - (1 - f(s, a)) \cdot r(s, a),$$

where the last step follows from the fact that $r(s, a) \leq 1$.

UPPER BOUNDING $\widehat{J}(\pi) - J(\pi)$. Next, using the upper and lower bounds on the reward bias, we now derive an upper bound on the difference between the value of a policy computed under the empirical MDP and the actual MDP. To compute this difference, we follow the following steps

$$\widehat{J}(\pi) - J(\pi) = \frac{1}{1 - \gamma} \sum_{s,a} \left(\widehat{d}^{\pi}_{\mathcal{D}^{\text{eff}}}(s) \pi(a|s) \widehat{r}^{\text{eff}}(s, a) - d^{\pi}(s) \pi(a|s) r(s, a) \right) \tag{E.5.5}$$

$$= \frac{1}{1 - \gamma} \underbrace{\sum_{s,a} \widehat{d}^{\pi}_{\mathcal{D}^{\text{eff}}}(s) \pi(a|s) \left(\widehat{r}^{\text{eff}}(s, a) - r(s, a) \right)}_{:= \Delta_1} + \frac{1}{1 - \gamma} \underbrace{\sum_{s,a} \left(\widehat{d}^{\pi}_{\mathcal{D}^{\text{eff}}}(s) - d^{\pi}(s) \right) \pi(a|s) r(s, a)}_{:= \Delta_2}$$

Following Kumar et al. [181] (Theorem 3.6), we can bound the second term Δ_2 using:

$$|\Delta_2| \leq \frac{\gamma C_{P,\delta}}{1 - \gamma} \mathbb{E}_{s \sim \widehat{d}^{\pi}_{\mathcal{D}^{\text{eff}}}(s)} \left[\frac{\sqrt{|\mathcal{A}|}}{\sqrt{|\mathcal{D}^{\text{eff}}(s)|}} \sqrt{D(\pi, \widehat{\pi}^{\text{eff}}_{\beta})(s) + 1} \right]. \tag{E.5.6}$$

To upper bound Δ_1, we utilize the reward upper bound from Equation E.5.2:

$$\Delta_1 = \sum_{\boldsymbol{s}} \widehat{d}^{\pi}_{\mathcal{D}^{\text{eff}}}(\boldsymbol{s}) \left(\sum_{\boldsymbol{a}} \pi(\boldsymbol{a}|\boldsymbol{s}) \left(\widehat{r}^{\text{eff}}(\boldsymbol{s}, \boldsymbol{a}) - r(\boldsymbol{s}, \boldsymbol{a}) \right) \right) \tag{E.5.7}$$

$$\leq \underbrace{\sum_{\boldsymbol{s}} \widehat{d}^{\pi}_{\mathcal{D}^{\text{eff}}}(\boldsymbol{s}) \sum_{\boldsymbol{a}} f(\boldsymbol{s},\boldsymbol{a}) \frac{C_{r,\delta}}{\sqrt{|\mathcal{D}(\boldsymbol{s})|}} \frac{\pi(\boldsymbol{a}|\boldsymbol{s})}{\sqrt{\hat{\pi}_\beta(\boldsymbol{a}|\boldsymbol{s})}}}_{=\Delta_1'} \underbrace{- \sum_{\boldsymbol{s},\boldsymbol{a}} \widehat{d}^{\pi}_{\mathcal{D}^{\text{eff}}}(\boldsymbol{s}) \pi(\boldsymbol{a}|\boldsymbol{s}) \left[(1 - f(\boldsymbol{s}, \boldsymbol{a})) \cdot r(\boldsymbol{s}, \boldsymbol{a}) \right]}_{:=\Delta_4}. \tag{E.5.8}$$

Combining the results so far, we obtain, for any policy π:

$$J(\pi) \geq \widehat{J}(\pi) - \frac{|\Delta_2|}{1-\gamma} - \frac{|\Delta_1'|}{1-\gamma} + \frac{\Delta_4}{1-\gamma}. \tag{E.5.9}$$

LOWER BOUNDING $\widehat{J}(\pi) - J(\pi)$. To lower bound this quantity, we follow the step shown in Equation E.5.5, and lower bound the term Δ_2 by using the negative of the RHS of Equation E.5.6, and lower bound Δ_1 by upper bounding its absolute value as shown below:

$$\Delta_1 = \sum_{\boldsymbol{s}} \widehat{d}^{\pi}_{\mathcal{D}^{\text{eff}}}(\boldsymbol{s}) \left(\sum_{\boldsymbol{a}} \pi(\boldsymbol{a}|\boldsymbol{s}) \left(\widehat{r}^{\text{eff}}(\boldsymbol{s}, \boldsymbol{a}) - r(\boldsymbol{s}, \boldsymbol{a}) \right) \right) \tag{E.5.10}$$

$$\geq \underbrace{\sum_{\boldsymbol{s}} \widehat{d}^{\pi}_{\mathcal{D}^{\text{eff}}}(\boldsymbol{s}) \sum_{\boldsymbol{a}} f(\boldsymbol{s},\boldsymbol{a}) \frac{C_{r,\delta}}{\sqrt{|\mathcal{D}(\boldsymbol{s})|}} \frac{\pi(\boldsymbol{a}|\boldsymbol{s})}{\sqrt{\hat{\pi}_\beta(\boldsymbol{a}|\boldsymbol{s})}}}_{=\Delta_1'} + \sum_{\boldsymbol{s}} \widehat{d}^{\pi}_{\mathcal{D}^{\text{eff}}}(\boldsymbol{s}) \sum_{\boldsymbol{a}} \pi(\boldsymbol{a}|\boldsymbol{s}) \cdot (1 - f(\boldsymbol{s}, \boldsymbol{a})) r(\boldsymbol{s}, \boldsymbol{a}). \tag{E.5.11}$$

This gives rise to the complete lower bound:

$$\widehat{J}(\pi) \geq J(\pi) - \frac{|\Delta_2|}{1-\gamma} - \frac{1}{1-\gamma} \sum_{\boldsymbol{s},\boldsymbol{a}} \widehat{d}^{\pi}_{\mathcal{D}_i^{\text{eff}}}(\boldsymbol{s}) \pi(\boldsymbol{a}|\boldsymbol{s}) (1 - f(\boldsymbol{s}, \boldsymbol{a})) r(\boldsymbol{s}, \boldsymbol{a}) - \frac{\Delta_1'}{1-\gamma}. \tag{E.5.12}$$

POLICY IMPROVEMENT TERM (II). Finally, the missing piece that needs to be bounded is the policy improvement term (ii) in the decomposition of $J(\pi) - J(\pi_\beta)$. Utilizing the abstract form of offline RL, we note that term (ii) is lower bounded as:

$$\text{term (ii)} \geq \frac{\alpha}{1-\gamma} D(\pi, \pi_\beta). \tag{E.5.13}$$

PUTTING IT ALL TOGETHER. To obtain the final expression of Proposition 7.3.1, we put all the parts together, and include some simplifications to obtain the final expression. The bound we show is relative to the effective behavior policy π_β^{eff}. Applying Equation E.5.12

for term (i) on policy π, Equation E.5.13 for term (ii), and Equation E.5.9 for the behavior policy π_β^{eff}, we obtain the following:

$$
\begin{aligned}
&J(\pi) - J(\pi_\beta^{\text{eff}}) = J(\pi) - \widehat{J}(\pi) + \widehat{J}(\pi) - \widehat{J}(\pi_\beta^{\text{eff}}) + \widehat{J}(\pi_\beta^{\text{eff}}) - J(\pi_\beta^{\text{eff}}) \\
&\geq -\frac{2\gamma C_{P,\delta}}{(1-\gamma)^2}\mathbb{E}_{\boldsymbol{s}\sim\widehat{d}^{\pi}_{\mathcal{D}^{\text{eff}}}(\boldsymbol{s})}\left[\frac{\sqrt{|\mathcal{A}|}}{\sqrt{|\mathcal{D}^{\text{eff}}(\boldsymbol{s})|}}\sqrt{D(\pi, \widehat{\pi}_\beta^{\text{eff}})(\boldsymbol{s}) + 1}\right] - \frac{2C_{r,\delta}}{1-\gamma}\mathbb{E}_{\boldsymbol{s},\boldsymbol{a}\sim\widehat{d}^{\pi}_{\mathcal{D}^{\text{eff}}}}\left[\frac{f(\boldsymbol{s},\boldsymbol{a})}{\sqrt{|\mathcal{D}(\boldsymbol{s},\boldsymbol{a})|}}\right] \\
&\quad - \frac{1}{1-\gamma}\left(\mathbb{E}_{\boldsymbol{s},\boldsymbol{a}\sim d^{\pi_\beta^{\text{eff}}}_{\mathcal{D}^{\text{eff}}}}\left[(1 - f(\boldsymbol{s},\boldsymbol{a}))\, r(\boldsymbol{s},\boldsymbol{a})\right]\right) + \frac{1}{1-\gamma}\mathbb{E}_{\boldsymbol{s},\boldsymbol{a}\sim\widehat{d}^{\pi}_{\mathcal{D}^{\text{eff}}}}\left[(1 - f(\boldsymbol{s},\boldsymbol{a}))\cdot r(\boldsymbol{s},\boldsymbol{a})\right] \\
&\quad + \frac{\alpha}{1-\gamma}D(\pi, \pi_\beta^{\text{eff}}).
\end{aligned}
$$

Note that in the second step above, we upper bound the quantities Δ_1' and Δ_2 corresponding to π_β^{eff} with twice the expression for policy π. This is because the effective behavior policy π_β^{eff} consists of a mixture of the original behavior policy $\hat{\pi}_\beta$ with the additional data, and thus the new effective dataset consists of the original dataset $\mathcal{D}_i$ as its part. Upper bounding it with twice the corresponding term for π is a valid bound, though a bit looser, but this bound suffices for our interpretations.

For our analysis purposes, we will define the suboptimality induced in the bound due to reward bias for a given u and v as:

$$
\begin{aligned}
&\text{RewardBias}(\pi, \pi_\beta^{\text{eff}}) && \text{(E.5.14)} \\
&= -\frac{1}{1-\gamma}\left[\mathbb{E}_{\boldsymbol{s},\boldsymbol{a}\sim\widehat{d}^{\pi}_{\mathcal{D}^{\text{eff}}}}\left[(1 - f(\boldsymbol{s},\boldsymbol{a}))\cdot r(\boldsymbol{s},\boldsymbol{a})\right] - \left(\mathbb{E}_{\boldsymbol{s},\boldsymbol{a}\sim d^{\pi_\beta^{\text{eff}}}_{\mathcal{D}^{\text{eff}}}}\left[(1 - f(\boldsymbol{s},\boldsymbol{a}))\, r(\boldsymbol{s},\boldsymbol{a})\right]\right)\right] && \text{(E.5.15)}
\end{aligned}
$$

Thus, we obtain the desired bound in Proposition 7.3.1. □

E.5.2 *When is Reward Bias Small? Proof of Theorem 7.3.2*

Next, we wish to understand when the reward bias in Equation E.5.14 is small. Concretely, we wish to search for effective behavior policies such that the dataset induced by them attains a small reward bias. Therefore we provide a proof for Theorem 7.3.2 in this section.

Proof. We can express the reward bias as:

$$
\text{RewardBias}(\pi, \pi_\beta^{\text{eff}}) := -\frac{1}{1-\gamma}\sum_{\boldsymbol{s},\boldsymbol{a}}\left(\widehat{d}^{\pi}_{\mathcal{D}^{\text{eff}}}(\boldsymbol{s},\boldsymbol{a}) - \widehat{d}^{\pi_\beta^{\text{eff}}}_{\mathcal{D}^{\text{eff}}}(\boldsymbol{s},\boldsymbol{a})\right)\cdot(1 - f(\boldsymbol{s},\boldsymbol{a}))\cdot r(\boldsymbol{s},\boldsymbol{a}),
$$

Now, since our goal is to minimize the reward bias with respect to the effective behavior policy, we minimize the expression for suboptimality induced due to reward bias, shown above with respect to π_β^{eff}. Before performing the differentiation step, we note the following simplification (we drop the $\mathcal{D}^{\text{eff}}$ from the notation in $d_{\mathcal{D}^{\text{eff}}}^{\pi_\beta{}^{\text{eff}}}$ to make the notation less cluttered below):

$$\min_{\pi_\beta^{\text{eff}}} \ \text{RewardBias}(\pi, \pi_\beta^{\text{eff}}) \tag{E.5.16}$$

$$:= \min_{\pi_\beta^{\text{eff}}} \ -\left(\widehat{J}(\pi) - \widehat{J}(\pi_\beta^{\text{eff}})\right) + \frac{1}{(1-\gamma)} \sum_{\boldsymbol{s},\boldsymbol{a}} r(\boldsymbol{s},\boldsymbol{a}) f(\boldsymbol{s},\boldsymbol{a}) \left(\widehat{d}_{\mathcal{D}^{\text{eff}}}^{\pi}(\boldsymbol{s},\boldsymbol{a}) - d_{\mathcal{D}^{\text{eff}}}^{\pi_\beta{}^{\text{eff}}}(\boldsymbol{s},\boldsymbol{a})\right)$$

$$= \min_{\pi_\beta^{\text{eff}}} \ -\widehat{J}(\pi) + \widehat{J}(\pi_\beta^{\text{eff}}) + \frac{1}{(1-\gamma)|\mathcal{D}^{\text{eff}}|} \sum_{\boldsymbol{s},\boldsymbol{a}} |\mathcal{D}(\boldsymbol{s},\boldsymbol{a})| r(\boldsymbol{s},\boldsymbol{a}) \left(\frac{\widehat{d}^{\pi}(\boldsymbol{s},\boldsymbol{a})}{\widehat{d}^{\pi_\beta^{\text{eff}}}(\boldsymbol{s},\boldsymbol{a})} - 1\right)$$

$$= \min_{\pi_\beta^{\text{eff}}} \ \widehat{J}(\pi_\beta^{\text{eff}}) + \frac{1}{(1-\gamma)|\mathcal{D}^{\text{eff}}|} \sum_{\boldsymbol{s},\boldsymbol{a}} |\mathcal{D}(\boldsymbol{s},\boldsymbol{a})| r(\boldsymbol{s},\boldsymbol{a}) \left(\frac{\widehat{d}^{\pi}(\boldsymbol{s},\boldsymbol{a})}{\widehat{d}^{\pi_\beta^{\text{eff}}}(\boldsymbol{s},\boldsymbol{a})} - 1\right). \tag{E.5.17}$$

Now, since we can express the entire objective in Equation E.5.17 as a function of $\widehat{d}^{\pi_\beta{}^{\text{eff}}}$ since $\widehat{J}(\pi_\beta^{\text{eff}}) = \sum_{\boldsymbol{s},\boldsymbol{a}} \widehat{d}^{\pi_\beta{}^{\text{eff}}}(\boldsymbol{s},\boldsymbol{a}) r(\boldsymbol{s},\boldsymbol{a})$, we can compute and set the derivative of Equation E.5.17 with respect to $\widehat{d}^{\pi_\beta{}^{\text{eff}}}(\boldsymbol{s},\boldsymbol{a})$ as o while adding down the constraints that pertain to the validity of π. This gives us:

$$\widehat{d}^{\pi}(\boldsymbol{s},\boldsymbol{a}) \propto \sqrt{\frac{|\mathcal{D}(\boldsymbol{s},\boldsymbol{a})| \cdot \widehat{d}^{\pi}(\boldsymbol{s},\boldsymbol{a})}{|\mathcal{D}^{\text{eff}}|}}.$$

Substituting $|\mathcal{D}(\boldsymbol{s},\boldsymbol{a})| = \widehat{d}_{\text{L}}(\boldsymbol{s},\boldsymbol{a}) \cdot |\mathcal{D}_{\text{L}}|$ we obtain the desired result. □

E.5.3 *When is The Bound in Theorem 7.3.1 Tightest?: Proof of Theorem 7.3.3*

In this section, we will formally state and provide a proof for Theorem 7.3.3. To prove this result, we first compute an upper bound on the error terms (a) and (b) in Theorem 7.3.3, and show that the optimized distribution shown in Theorem 7.3.3 emerges as a direct consequence of optimizing this bound. To begin, we compute a different upper bound on sampling error than the one used in Theorem 7.3.3. Defining the sampling error for a policy π as the difference in return in the original and the empirical MDPs $\Delta_{\text{sampling}} = \widehat{J}(\pi) - J(\pi)$, we obtain the following Lemma:

Lemma E.5.2 (Upper bound on sampling error in terms of $\widehat{d}^{\pi}(\boldsymbol{s},\boldsymbol{a})$)**.** *We can upper bound sampling error term as follows:*

$$\Delta_{sampling} \leq \frac{\gamma C_{P,\delta}}{(1-\gamma)^2 \sqrt{|\mathcal{D}|}} \sum_{\boldsymbol{s},\boldsymbol{a}} \frac{\widehat{d}^{\pi}(\boldsymbol{s},\boldsymbol{a})}{\sqrt{\widehat{d}^{\pi_\beta^{\text{eff}}}(\boldsymbol{s},\boldsymbol{a})}}. \tag{E.5.18}$$

Proof. For this proof, we will derive a bound on the sampling error, starting from scratch, but this time only in terms of the state-action marginals:

$$\Delta_{\text{sampling}} = \frac{1}{1-\gamma} \sum_{\boldsymbol{s},\boldsymbol{a}} \left(\widehat{d}^{\pi}(\boldsymbol{s},\boldsymbol{a}) - d^{\pi}(\boldsymbol{s},\boldsymbol{a}) \right) \cdot r(\boldsymbol{s},\boldsymbol{a})$$

Note that we can bound Δ_{sampling} by upper bounding the total variation between marginal state-action distributions in the empirical and actual MDPs, i.e., $||\widehat{d}^{\pi} - d^{\pi}||_1$, since $|r_M(\boldsymbol{s},\boldsymbol{a})| \leq R_{\max}$. and hence we bound the second term effectively. Our analysis is similar to Achiam et al. [3]. Define, $G = (I - \gamma P^{\pi})^{-1}$ and $\boldsymbol{a}rG = (I - \gamma \widehat{P}^{\pi})^{-1}$. Then,

$$\widehat{d}^{\pi} - d^{\pi} = (1-\gamma)(\boldsymbol{a}rG - G)\rho,$$

where $\rho(\boldsymbol{s})$ is the initial state distribution. Then we can use the derivation in proof of Theorem 3.6 in Kumar et al. [181] or Equation 21 from Achiam et al. [3], to bound this difference as

$$\begin{aligned} ||d^{\pi} - \widehat{d}^{\pi}||_1 &\leq \frac{\gamma}{1-\gamma} \sum_{\boldsymbol{s}} \widehat{d}^{\pi}(\boldsymbol{s}) \frac{C_{P,\delta}}{\sqrt{|\mathcal{D}(\boldsymbol{s})|}} \sum_{\boldsymbol{a}} \frac{\pi(\boldsymbol{a}|\boldsymbol{s})}{\sqrt{\pi_{\beta}(\boldsymbol{a}|\boldsymbol{s})}} \\ &\leq \frac{\gamma C_{P,\delta}}{(1-\gamma)\sqrt{|\mathcal{D}|}} \sum_{\boldsymbol{s},\boldsymbol{a}} \frac{\widehat{d}^{\pi}(\boldsymbol{s},\boldsymbol{a})}{\sqrt{\widehat{d}^{\pi^{\text{eff}}_{\beta}}(\boldsymbol{s},\boldsymbol{a})}}. \end{aligned}$$

Thus the sampling error can be bounded by:

$$\Delta_{\text{sampling}} \leq \frac{\gamma C_{P,\delta}}{(1-\gamma)^2\sqrt{|\mathcal{D}|}} \sum_{\boldsymbol{s},\boldsymbol{a}} \frac{\widehat{d}^{\pi}(\boldsymbol{s},\boldsymbol{a})}{\sqrt{\widehat{d}^{\pi^{\text{eff}}_{\beta}}(\boldsymbol{s},\boldsymbol{a})}},$$

which proves the lemma. □

Theorem E.5.3 (Optimized reweighting unlabeled data). *The optimal effective behavior policy that maximizes a lower bound on $J(\pi^{\text{eff}}_{\beta}) - [(a) + (b)]$ in Theorem 7.3.1 satisfies: $\widehat{d}^{\pi^{\text{eff}}_{\beta}}(\boldsymbol{s},\boldsymbol{a}) = p^*(\boldsymbol{s},\boldsymbol{a})$, where,*

$$p^* = \arg \min_{p \in \Delta^{|\mathcal{S}||\mathcal{A}|}} \sum_{\boldsymbol{s},\boldsymbol{a}} C_1 \frac{\widehat{d}^{\pi}(\boldsymbol{s},\boldsymbol{a})}{\sqrt{p(\boldsymbol{s},\boldsymbol{a})}} + C_2 |d_{\text{L}}(\boldsymbol{s},\boldsymbol{a})| \frac{\widehat{d}^{\pi}(\boldsymbol{s},\boldsymbol{a})}{p(\boldsymbol{s},\boldsymbol{a})},$$

where C_1 and C_2 are universal positive constants that depend on the sizes of the labeled and unlabeled datasets are shown in Equation E.5.19.

Proof. To prove this result, we will first simplify the expression containing all terms except the policy imnprovement term in Theorem 7.3.3, so that we can then maximize the bound to obtain the statement of our theoretical statement.

$$
\begin{aligned}
(\bullet) &:= J(\pi_\beta^{\text{eff}}) - [(a) + (b)] \geq \\
&J(\pi_\beta^{\text{eff}}) + \frac{1}{1-\gamma} \sum_{s,a} \left(\widehat{d}^{\pi}(s,a) - \widehat{d}^{\pi_\beta{}^{\text{eff}}}(s,a) \right) \cdot (1 - f(s,a)) \cdot r(s,a) - \Delta_{\text{sampling}} \\
&\geq J(\pi_\beta^{\text{eff}}) + \frac{1}{1-\gamma} \sum_{s,a} \left(\widehat{d}^{\pi}(s,a) - \widehat{d}^{\pi_\beta{}^{\text{eff}}}(s,a) \right) \cdot (1 - f(s,a)) \cdot r(s,a) \\
&\qquad - \frac{\gamma C_{P,\delta}}{(1-\gamma)^2 \sqrt{|\mathcal{D}_{\text{eff}}|}} \sum_{s,a} \frac{\widehat{d}^{\pi}(s,a)}{\sqrt{\widehat{d}^{\pi_\beta^{\text{eff}}}(s,a)}}
\end{aligned}
$$

First of all we can lower bound, $J(\pi_\beta^{\text{eff}})$ in terms of the return of $\pi_\beta^{\text{eff}})$ in the empirical MDP induced by the dataset $\mathcal{D}_{\text{eff}}$ and an irreducible sampling error term, that grows as $\mathcal{O}\left(\sqrt{1/|\mathcal{D}^{\text{eff}}|}\right)$ and does not depend on the distribution of state-action pairs in the effective dataset, $\widehat{d}^{\pi_\beta^{\text{eff}}}(s,a)$, but only depends on its size. Using this result, and by performing algebraic manipulation, we can further simplify this as:

$$
\begin{aligned}
(\bullet) &\geq \widehat{J}(\pi_\beta^{\text{eff}}) + \widehat{J}(\pi) - \widehat{J}(\pi_\beta^{\text{eff}}) - \frac{\sum_{s,a} |\mathcal{D}(s,a)| r(s,a) \left(\frac{\widehat{d}^{\pi}(s,a)}{\widehat{d}^{\pi_\beta^{\text{eff}}}(s,a)} - 1 \right)}{(1-\gamma)|\mathcal{D}^{\text{eff}}|} \\
&\qquad - \frac{\gamma C_{P,\delta}}{(1-\gamma)^2 \sqrt{|\mathcal{D}_{\text{eff}}|}} \sum_{s,a} \frac{\widehat{d}^{\pi}(s,a)}{\sqrt{\widehat{d}^{\pi_\beta^{\text{eff}}}(s,a)}} + \mathcal{O}\left(\sqrt{\frac{1}{|\mathcal{D}^{\text{eff}}|}} \right) \\
&\geq \widehat{J}(\pi) - \frac{1}{(1-\gamma)|\mathcal{D}^{\text{eff}}|} \sum_{s,a} |\mathcal{D}(s,a)| r(s,a) \left(\frac{\widehat{d}^{\pi}(s,a)}{\widehat{d}^{\pi_\beta^{\text{eff}}}(s,a)} - 1 \right) \\
&\qquad - \frac{\gamma C_{P,\delta}}{(1-\gamma)^2 \sqrt{|\mathcal{D}_{\text{eff}}|}} \sum_{s,a} \frac{\widehat{d}^{\pi}(s,a)}{\sqrt{\widehat{d}^{\pi_\beta^{\text{eff}}}(s,a)}} + \mathcal{O}\left(\sqrt{\frac{1}{|\mathcal{D}^{\text{eff}}|}} \right) \\
&\geq \widehat{J}(\pi) - \frac{1}{(1-\gamma)|\mathcal{D}^{\text{eff}}|} \sum_{s,a} |\mathcal{D}(s,a)| \left(\frac{\widehat{d}^{\pi}(s,a)}{\widehat{d}^{\pi_\beta^{\text{eff}}}(s,a)} - 1 \right) \\
&\qquad - \frac{\gamma C_{P,\delta}}{(1-\gamma)^2 \sqrt{|\mathcal{D}_{\text{eff}}|}} \sum_{s,a} \frac{\widehat{d}^{\pi}(s,a)}{\sqrt{\widehat{d}^{\pi_\beta^{\text{eff}}}(s,a)}} + \mathcal{O}\left(\sqrt{\frac{1}{|\mathcal{D}^{\text{eff}}|}} \right),
\end{aligned}
$$

where the last inequality follows from the fact that $|r(s,a)| \leq 1$. Since $|\mathcal{D}^{\text{eff}}|$ and $\widehat{d}^{\pi_\beta^{\text{eff}}}$ are decoupled (one is the distribution; other is the size of the dataset), we can optimize over

Environment	Dataset type / size	CDS+UDS	UDS	No Sharing
Meta-World door open	expert / 2k transitions	**67.6%**	58.8%	31.3%
	medium / 2k transitions	67.3%	**74.2%**	27.6%
	medium-replay / 152k transitions	**30.0%**	0.0%	14.8%

Table 52: We perform an empirical analysis on the Meta-World door open task where we use varying data quality and dataset size target task door open. We share the same dataset from the other three tasks in the multi-task Meta-World environment, door close, drawer open and drawer close to the target task. The numbers are averaged over three random seeds. CDS+UDS and UDS are able to outperform No Sharing in most of the settings except that UDS fails to achieve non-zero success rate in the medium-replay dataset with a large number of transitions. Such results suggest that CDS+UDS and UDS are robust to the data quality of the target task and work the best in settings where the target task has limited data.

each of them independently, and hence, we find that the distribution that optimizes this bound is given by:

$$p^* = \arg \min_{p \in \Delta^{|\mathcal{S}||\mathcal{A}|}} \sum_{\boldsymbol{s},\boldsymbol{a}} C_1 \frac{\widehat{d}^{\pi}(\boldsymbol{s},\boldsymbol{a})}{\sqrt{p(\boldsymbol{s},\boldsymbol{a})}} + C_2 |d_{\mathrm{L}}(\boldsymbol{s},\boldsymbol{a})| \frac{\widehat{d}^{\pi}(\boldsymbol{s},\boldsymbol{a})}{p(\boldsymbol{s},\boldsymbol{a})},$$

where:

$$C_2 := \frac{|\mathcal{D}_{\mathrm{L}}|}{(1-\gamma)|\mathcal{D}_{\mathrm{eff}}|}, \qquad C_1 := \frac{\gamma C_{P,\delta}}{(1-\gamma)^2 \sqrt{|\mathcal{D}_{\mathrm{eff}}|}}. \tag{E.5.19}$$

This proves Theorem 7.3.3. □

E.6 Additional Empirical Analysis

In this section, we perform an empirical study on the Meta-World domain to better understand the reason that UDS and CDS+UDS work well. Our theoretical analysis suggests that UDS will help the most on domains with limited data or narrow coverage or low data quality. To test these conditions in practice, we perform empirical analysis on two domains as follows.

E.6.1 *Meta-World Domains*

We first choose the door open task with three different combinations of dataset size and data quality of the task-specific data with reward labels:

- 2k transitions with the expert-level performance (i.e. **high-quality data with limited sample size and narrow coverage**)
- 2k transitions with medium-level performance (i.e. **medium-quality data with limited sample size and narrow coverage**)

Environment	Labeled dataset type / size	Unlabeled dataset type / size	CDS+UDS	UDS	Sharing All (oracle)
D4RL hopper	random / 10k transitions	expert / 10k transitions	**10.1**	10.0	71.9
	random / 10k transitions	expert / 100k transitions	**105.8**	81.8	96.3
	random / 10k transitions	expert / 1M transitions	**102.3**	97.0	102.8

Table 53: Ablation study on the unlabeled dataset size ranging from 10k to 1M transitions in the single-task hopper domain. We bold the best method without true reward relabeling.

- a medium-replay dataset with 152k transitions (i.e. **medium-quality data with sufficient sample size and broad coverage**).

We share the same data from the other three tasks, door close, drawer open and drawer close as in Table 18. As shown in Table 52, both UDS and CDS+UDS are able to outperform No Sharing in the three settings, suggesting that increasing the coverage of the offline data as suggested by our theory does lead to performance boost in wherever we have limited good-quality data (expert), limited medium-quality data (medium) and abundant medium-quality data (medium-replay). It's worth noting that UDS and CDS+UDS significantly outperform No Sharing in the limited expert and medium data setting whereas in the medium-replay setting with broader coverage, CDS+UDS outperforms No sharing but UDS fails to achieve non-zero success rate. Such results suggest that UDS and CDS+UDS can yield greater benefit when the target task doesn't have sufficient data and the number of relabeled data is large. The fact that UDS is unable to learn on medium-replay datasets also suggests that data sharing without rewards is less useful in settings where the coverage of the labeled offline data is already quite broad.

E.6.2 *Ablation: Dataset Size*

Following the discussion in Section 7.3.4, we further study the setting where relabeled data has higher quality than the labeled data in the single-task hopper task by varying the amount of unlabeled data within the range of (10k, 100k, 1M) transitions. We pick the case where labeled data is random and unlabeled data is expert for such ablation study. As shown in Table 53, as the unlabeled (expert) dataset size decreases, the result of UDS drops significantly whereas Sharing All retains a reasonable level of performance. This ablation suggests that as the effective dataset size decreases, the benefit of reducing sampling error is reduced and no longer able to outweigh the reward bias as indicated in our theoretical analysis. Moreover, Table 53 also suggests that, in the setting with medium unlabeled dataset size (100k transitions), CDS+UDS is able to prevent the performance drop as seen in UDS and even outperforms Sharing All. However, CDS+UDS cannot successfully tackle the case where there are only 10k unlabeled transitions, suggesting that the reward bias optimized by CDS+UDS is still detrimental in the limited unlabeled data regime.

Env	Labeled dataset type / size	Unlabeled dataset type / size	CDS+UDS	UDS	Reward Pred.
hopper	expert / 10k transitions	medium / 1M transitions	**78.3**± 5.4	64.4±11.7	51.7 ±8.4
	expert / 20k transitions	medium / 1M transitions	95.5±5.4	**99.2**±3.1	96.7±3.6
	random / 10k transitions	medium / 1M transitions	**65.8**±11.3	51.9±2.4	33.4±5.4
	random / 20k transitions	medium / 1M transitions	**69.1**±4.8	59.9±5.6	47.5±5.9

Table 54: Ablation study on comparisons between CDS+UDS/UDS and Reward Predictor with varying labeled dataset size and quality in the single-task hopper domain. We bold the best method.

E.6.3 *Comparison to Reward-Learning Methods*

We performed an ablation that varies the labeled data size (10k & 20k transitions) and quality (expert / random) for reward learning methods on D4RL hopper, with unlabeled data being 1M hopper-medium transitions. Observe on the right, as expected, reward learning performs better with more labeled data. Furthermore, while reward learning works well when labeled data is high quality, it fails to perform well when labeled data is of low quality, potentially due to the bias in reward prediction. UDS and CDS+UDS are less sensitive to labeled data quality/size.

E.6.4 *Takeaways from the empirical analysis*

Given our empirical analysis in Table 17, Table 52, Table 53 and Table 54, we summarize the applicability of UDS / CDS+UDS under different scenarios for practitioners in Figure 38 below.

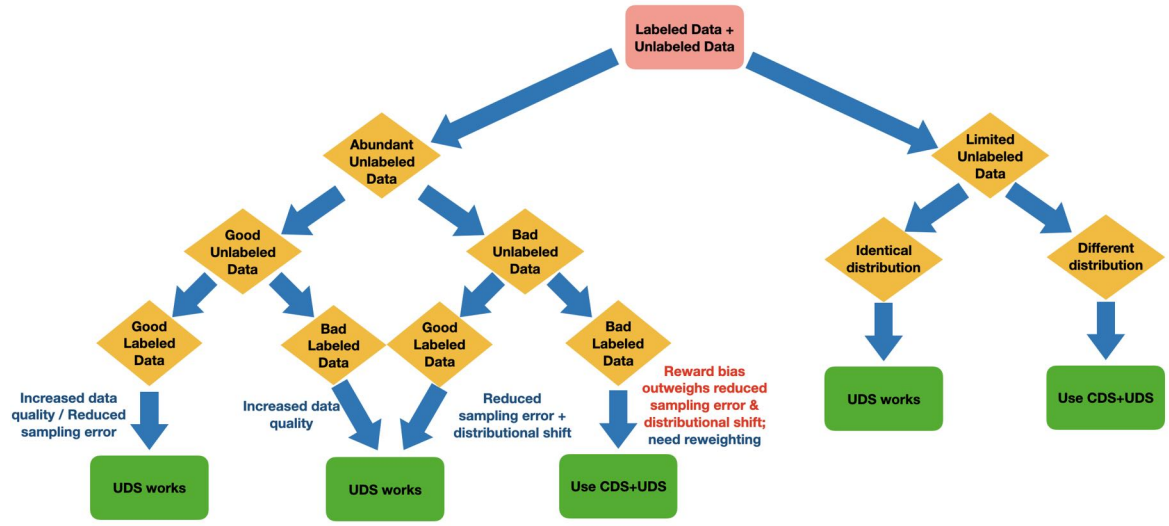


Figure 38: A tree plot that illustrates under which conditions a practitioner should run UDS over apply the optimized reweighting scheme on top of UDS.

Environment	Tasks	Oracle Success Rate of the Shared data
Meta-World	drawer open	47.4%
	door close	99.2%
	drawer open	0.1%
	drawer close	91.6%%
	average	59.5%
AntMaze	medium maze (3 tasks) average	4.3%
	large maze (7 tasks) average	1.6%

Table 55: Success rate of the data shared from other tasks to the target task determined by the ground-truth multi-task reward function.

E.7 ADDITIONAL DETAILS ABOUT DATA QUALITY

We present the success rate of the data shared from other tasks to the target task computed by the oracle multi-task reward function in both the multi-task Meta-World and AntMaze domains in Table 55. Note that the success rate of drawer close and door close are particularly high since for other tasks, the drawer / door is initialized to be closed and therefore the success rate of other task data for these two tasks are almost 100% as defined by the success condition in the public Meta-World repo. Apart from these two particularly high success rates, the success rates of the shared data are consistently above 0% across all tasks in both domains. This fact suggests that UDS and CDS+UDS are *not* relabeling with the ground truth reward where the relabeled data are actually all failures but rather performs the conservative bellman backups on relabeled data that is shown to be effective empirically.

To better understand the performance of UDS under different relabeled data quality, we evaluate the UDS under different success rates of the data relabeled from other tasks in the multi-task Meta-World domain. Specifically, we filter out data shared from other tasks to ensure that the success rates of the relabeled data are 5%, 50% and 90% respectively. We compare the results of UDS on such data compositions to the performance of UDS in Table 18 where the success rate of relabeled data is 59.6% as shown in Table 55. The full results are in Table 56. UDS on relabeled data with 50% and 90% success rates achieves similar results compared to original UDS whereas UDS on relabel data with 5% success rate is significantly worse. Hence, UDS can obtain good results in settings where the relabeled data is of high quality despite incurring high reward bias, but is not helpful in settings where the shared data is of low quality and does not offer much information about solving the target task.

E.8 ADDITIONAL EMPIRICAL RESULTS FOR UDS AND CDS+UDS

In this section, we evaluate UDS and CDS+UDS in the multi-task locomotion setting with dense rewards. We pick the multi-task walker environment as used in prior work [366], which consists of three tasks, run forward, run backward and jump. The reward functions

Environment	Tasks	UDS	UDS-5% relabel success	UDS-50% relabel success	UDS-90% relabel success
Meta-World	drawer open	51.9%±25.3	0.0%±0.0%	57.3%±18.9%	73.3%±8.6%
	door close	12.3%±27.6%	0.0%±0.0%	0.0%±0.0%	0.0%±0.0%
	drawer open	61.8%±16.3%	19.4%±27.3%	61.0%±12.7%	56.3%±20.3%
	drawer close	99.6%±0.7%	66.0%±46.7%	99.7%±0.5%	100.0%±0.0%
	average	56.4%±12.8%	21.4% ±16.1%	54.3% ±2.0%	57.4%±3.3%

Table 56: Performance of UDS under different actual success rates of the relabeled data.

Environment	Tasks / Dataset type	CDS+UDS	UDS	No Sharing	Reward Predictor	CDS (oracle)	Sharing All (oracle)
Multi-Task Walker	run forward / medium-replay	**880.1**±108.8	665.0±84.9	590.1±48.6	520.7±373.6	1057.9±121.6	701.4±47.0
	run backward / medium	**717.8**±78.3	689.3±16.3	614.7±87.3	417.3±235.3	564.8±47.7	756.7±76.7
	jump / expert	1487.7±177.6	1036.0±247.1	**1575.2**±70.9	583.0±432.0	1418.2±138.4	885.1±152.9
	average	**1028.6**±76.8	796.7±106.3	926.6±37.7	506.7 ± 343.6	1013.6±71.5	781.0±100.8

Table 57: Results for multi-task walker experiment with dense rewards. We only bold the best-performing method that does not have access to the true reward during relabeling. CDS+UDS and UDS are able to outperform No Sharing and Reward Predictor while attaining competitive results compared to CDS and Sharing All with oracle rewards.

of the three tasks are $r(s,a) = v_x - 0.001 * \|a\|_2^2$, $r(s,a) = -v_x - 0.001 * \|a\|_2^2$ and $r(s,a) = -\|v_x\| - 0.001 * \|a\|_2^2 + 10 * (z - \text{init z})$ respectively where v_x denotes the velocity along the x-axis and z denotes the z-position of the half-cheetah and init z denotes the initial z-position. In UDS and CDS+UDS, we relabel the rewards routed from other tasks with the minimum reward value in the offline dataset of the target task. As shown in Table 57, CDS+UDS and UDS outperform No Sharing and Reward Predictor by a large margin while also performing comparably to CDS and Sharing All. Therefore, CDS+UDS and UDS are able to solve multi-task locomotion tasks with dense rewards.

E.9 COMPARISONS OF CDS+UDS AND UDS TO MODEL-BASED RL

In this section, we compare CDS+UDS and UDS to a recent, state-of-the-art model-based offline RL method COMBO [367] in the Meta-World domain. We adapt COMBO to the multi-task offline setting by learning the dynamics model on data of all tasks combined and and performing vanilla multi-task offline training without data sharing using the model learned with all of the data. As shown in Table 58, CDS+UDS and UDS indeed outperform COMBO in the average task success rate. The intuition behind this is that COMBO is unable to learn an accurate dynamics model for tasks with limited data as in our Meta-World setting.

E.10 COMPARISONS TO PRE-TRAINED REPRESENTATION LEARNING

A popular strategy to utilize large amounts of unlabeled data along with some limited amount of labeled data in supervised learning is to utilize the unlabeled data for learning representations, and then running supervised training on the labeled data. A similar strategy has been utilized for RL and imitation learning in some prior work [354, 355].

Environment	Tasks	CDS+UDS	UDS	COMBO [367]
Meta-World	door open	**61.3%**±7.9%	51.9%±25.3%	0.0%±0.0%
	door close	**54.0%** ±42.5%	12.3%±27.6%	1.1%±1.6%
	drawer open	**73.5%**±9.6%	61.8%±16.3%	15.7%±15.2%
	drawer close	99.3%±0.7%	**99.6%**±0.7%	85.7%±13.3%
	average	**71.2%** ± 11.3%	56.4%±12.8%	25.6%±6.2%

Table 58: On the multi-task Meta-World domain, we compare CDS+UDS and UDS to the model-based offline RL method COMBO [367] that trains a dynamics model on all of the data and performs model-based offline training using the learned model. CDS+UDS and UDS are able to outperform COMBO by a large margin.

On the other hand, the UDS and CDS+UDS strategies take a different route of handling unlabeled data, and it is natural to wonder how these representation learning approaches compare to our approach, UDS, that runs offline RL with the lowest possible reward, with an optional reweighting scheme.

To investigate this, we run experiments comparing our UDS and CDS+UDS approaches to state-of-the-art representation learning approaches for utilizing unlabeled data. First, we compare UDS and CDS+UDS to the ACL [354] approach on the multi-task Meta-World domain. ACL pretrains a representation using a contrastive loss on both the labeled and unlabeled offline datasets and then runs standard multi-task offline RL using the pretrained representation with **No Sharing**. To handle the multi-task Meta-World domain, we use the version of ACL without inputting reward labels. ACL can be viewed as an alternative to our unlabeled sharing data scheme, which leverages unlabeled data for representation learning rather than sharing it directly. We show the comparison to ACL in Table 59. UDS and CDS+UDS outperform ACL in the average task performance while ACL is only proficient on drawer-open and drawer-close, and it cannot solve door-open or door-close. This indicates that sharing the unlabeled data, even when labeled with the minimum possible reward is important for offline RL performance, while pretraining representations on the whole multi-task offline dataset might have limited benefit. Also note that, in principle, UDS / CDS+UDS are complementary to ACL and these approaches can be combined together to further improve performance, which we leave as future work.

We also compare the results of UDS and CDS + UDS in the single-task AntMaze medium-play and large-play domains (shown in Table 16 in the main paper) to imitation learning methods, TRAIL [355] and other baselines from Yang et al. [355], that also utilize unlabeled data. These methods train a representation using the unlabeled dataset, and then run behavioral cloning to imitate the expert trajectories, which are limited in number. For example, the TRAIL method learns a state-conditioned action representation, by fitting an energy-based model to the transition dynamics of the unlabeled dataset, and then performs downstream imitation learning using the labeled expert dataset. In contrast to

Environment	Tasks	CDS + UDS (ours)	UDS (ours)	ACL
Meta-World	door open	**61.3%**±7.9%	51.9%±25.3%	2.8%±2.0%
	door close	54.0% ±42.5%	12.3%±27.6%	0.0%±0.0%
	drawer open	73.5%±9.6%	61.8%±16.3%	**83.2%**±14.2%
	drawer close	99.3%±0.7%	99.6%±0.7%	**100.0%**±0.0%
	average	**71.2%** ± 11.3%	56.4%±12.8%	46.4%±3.5%

Table 59: Comparison between **UDS** / **CDS+UDS** and the **ACL** [354] that performs representation learning on the unlabeled data instead of data sharing. Both **UDS** and **CDS+UDS** outperform **ACL** by a significant margin in the average task result, suggesting that sharing the unlabeled data is crucial in improving the performance of multi-task offline RL with unlabeled data compared to only using the data for learning the representation.

these prior approaches, UDS and CDS+UDS do not make any assumptions about how expert the labeled dataset is, and so they are not technically comparable to these imitation learning methods directly. Moreover, any specialized representation learning objective can also be combined with UDS and CDS+UDS to further improve performance. Nevertheless, we hope that a direct comparison to representation learning on unlabeled data combined with downstream imitation will provide informative insights about the potential of UDS and CDS+UDS to effectively leverage unlabeled data. Comparing Table 16 to Figure 3 in Yang et al. [355], we find that CDS+UDS outperforms the best method from Yang et al. [355], TRAIL, on the antmaze-medium-play task and performs a bit worse on the antmaze-large-play task. CDS+UDS also outperforms all the other methods in Figure 3 of [355]. Overall, this implies that UDS and CDS+UDS methods can perform comparably to state-of-the-art representation learning approaches with downstream imitation, without needing any specialized representation learning.

E.11 DETAILS OF UDS AND CDS+UDS

In this section, we include the details of training UDS and CDS+UDS in Appendix E.11.1 as well as details on the environment and datasets used in our experiments in Appendix E.11.2.

E.11.1 *Training Details*

We first present our practical implementation of UDS optimizes the following objectives for the Q-functions and the policy in the *single-task offline RL setting*:

$$\hat{Q}^{k+1} \leftarrow \arg\min_{\hat{Q}} \beta \left(\mathbb{E}_{\boldsymbol{s} \sim \mathcal{D}_{\mathrm{L}} \cup \mathcal{D}_{\mathrm{U}}, \boldsymbol{a} \sim \mu(\cdot|\boldsymbol{s})} \left[\hat{Q}(\boldsymbol{s}, \boldsymbol{a}) \right] - \mathbb{E}_{\boldsymbol{s}, \boldsymbol{a} \sim \mathcal{D}_{\mathrm{L}} \cup \mathcal{D}_{\mathrm{U}}} \left[\hat{Q}(\boldsymbol{s}, \boldsymbol{a}) \right] \right) + \frac{1}{2} \mathbb{E}_{(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{s}') \sim \mathcal{D}_{\mathrm{L}} \cup \mathcal{D}_{\mathrm{U}}} \left[\left(\hat{Q}(\boldsymbol{s}, \boldsymbol{a}) - \left(r(\boldsymbol{s}, \boldsymbol{a}) + \gamma Q(\boldsymbol{s}', \boldsymbol{a}') \right) \right)^2 \right],$$

and

$$\pi \leftarrow \arg\max_{\pi'} \mathbb{E}_{\boldsymbol{s} \sim \mathcal{D}_{\mathrm{L}} \cup \mathcal{D}_{\mathrm{U}}, \boldsymbol{a} \sim \pi'(\cdot|\boldsymbol{s})} \left[\hat{Q}^{\pi}(\boldsymbol{s}, \boldsymbol{a}) \right],$$

Moreover, CDS+UDS optimizes the following objectives for training the critic and the policy with a soft weight:

$$\hat{Q}^{k+1} \leftarrow$$
$$\arg\min_{\hat{Q}} \beta \left(\mathbb{E}_{\boldsymbol{s}\sim\mathcal{D}_{\mathrm{L}}\cup\mathcal{D}_{\mathrm{U}},\boldsymbol{a}\sim\mu(\cdot|\boldsymbol{s})} \left[w_{\mathrm{CDS}}(\boldsymbol{s},\boldsymbol{a};\mathrm{U}\rightarrow\mathrm{L})\hat{Q}(\boldsymbol{s},\boldsymbol{a}) \right] - \mathbb{E}_{\boldsymbol{s},\boldsymbol{a}\sim\mathcal{D}_{\mathrm{L}}\cup\mathcal{D}_{\mathrm{U}}} \left[w_{\mathrm{CDS}}(\boldsymbol{s},\boldsymbol{a};\mathrm{U}\rightarrow\mathrm{L})\hat{Q}(\boldsymbol{s},\boldsymbol{a}) \right] \right)$$
$$+ \frac{1}{2}\mathbb{E}_{(\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}')\sim\mathcal{D}_{\mathrm{L}}\cup\mathcal{D}_{\mathrm{U}}} \left[w_{\mathrm{CDS}}(\boldsymbol{s},\boldsymbol{a};\mathrm{U}\rightarrow\mathrm{L}) \left(\hat{Q}(\boldsymbol{s},\boldsymbol{a}) - \left(r(\boldsymbol{s},\boldsymbol{a}) + \gamma Q(\boldsymbol{s}',\boldsymbol{a}') \right) \right)^2 \right],$$

and

$$\pi \leftarrow \arg\max_{\pi'} \mathbb{E}_{\boldsymbol{s}\sim\mathcal{D}_{\mathrm{L}}\cup\mathcal{D}_{\mathrm{U}},\boldsymbol{a}\sim\pi'(\cdot|\boldsymbol{s})} \left[w_{\mathrm{CDS}}(\boldsymbol{s},\boldsymbol{a};\mathrm{U}\rightarrow\mathrm{L})\hat{Q}^{\pi}(\boldsymbol{s},\boldsymbol{a}) \right],$$

where β is the coefficient of the CQL penalty on distribution shift, μ is an action sampling distribution that covers the action bound as in CQL. On the hopper domain, when the unlabeled data is random, we use the version of CQL that does not maximize the term $\mathbb{E}_{\boldsymbol{s},\boldsymbol{a}\sim\mathcal{D}_{\mathrm{L}}\cup\mathcal{D}_{\mathrm{U}}} \left[\hat{Q}(\boldsymbol{s},\boldsymbol{a}) \right]$ to prevent overestimating Q-values on low-quality random data and use $\beta = 1.0$. We use $\beta = 5.0$ in the other settings in the hopper domain. On the AntMaze domain, following prior works [181, 366], we use the Lagrange version of CQL, where the coefficient β is automatically tuned against a pre-specified constraint value on the CQL loss equal to $\tau = 10.0$. We adopt other hyperparameters used in [366]. We adapt the CDS weight to the single-task setting as follows: $w_{\mathrm{CDS}}(\boldsymbol{s},\boldsymbol{a};\mathrm{U}\rightarrow\mathrm{L}) := \sigma\left(\frac{\Delta(\boldsymbol{s},\boldsymbol{a};\mathrm{U}\rightarrow\mathrm{L})}{\tau}\right)$ where $\Delta(\boldsymbol{s},\boldsymbol{a};\mathrm{U}\rightarrow\mathrm{L}) = \hat{Q}^{\pi}(\boldsymbol{s},\boldsymbol{a}) - P_{k\%}\{\hat{Q}^{\pi}(\boldsymbol{s}',\boldsymbol{a}')\colon \boldsymbol{s}',\boldsymbol{a}' \sim \mathcal{D}_{\mathrm{L}}\}$ for $(\boldsymbol{s},\boldsymbol{a}) \sim \mathcal{D}_{\mathrm{U}}$. We use $k = 50$ in all single-task domains.

Similarly in the *multi-task offline RL* setting, our practical implementation of UDS optimizes the following objectives for the Q-functions and the policy:

$$\begin{aligned}\hat{Q}^{k+1} \leftarrow \arg\min_{\hat{Q}} \mathbb{E}_{i\sim[N]} \Big[\beta \Big(& \mathbb{E}_{j\sim[N]} \left[\mathbb{E}_{\boldsymbol{s}\sim\mathcal{D}_j,\boldsymbol{a}\sim\mu(\cdot|\boldsymbol{s},i)} \left[\hat{Q}(\boldsymbol{s},\boldsymbol{a},i) \right] \right. \\ & \left. - \mathbb{E}_{\boldsymbol{s},\boldsymbol{a}\sim\mathcal{D}_j} \left[\hat{Q}(\boldsymbol{s},\boldsymbol{a},i) \right] \right] \Big) \\ & + \frac{1}{2}\mathbb{E}_{j\sim[N],(\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}')\sim\mathcal{D}_j} \left[\left(\hat{Q}(\boldsymbol{s},\boldsymbol{a},i) - \left(r(\boldsymbol{s},\boldsymbol{a},i)1\{\}\{j=i\} + \gamma Q(\boldsymbol{s}',\boldsymbol{a}') \right) \right)^2 \right] \Big],\end{aligned}$$

and

$$\pi \leftarrow \arg\max_{\pi'} \mathbb{E}_{i\sim[N]} \left[\mathbb{E}_{j\sim[N],\boldsymbol{s}\sim\mathcal{D}_j,\boldsymbol{a}\sim\pi'(\cdot|\boldsymbol{s},i)} \left[\hat{Q}^{\pi}(\boldsymbol{s},\boldsymbol{a},i) \right] \right],$$

We also present the objective of CDS+UDS in the multi-task task as follows:

$$\begin{aligned}\hat{Q}^{k+1} \leftarrow \arg\min_{\hat{Q}} \mathbb{E}_{i\sim[N]} \Big[\beta \Big(& \mathbb{E}_{j\sim[N]} \left[\mathbb{E}_{\boldsymbol{s}\sim\mathcal{D}_j,\boldsymbol{a}\sim\mu(\cdot|\boldsymbol{s},i)} \left[w_{\mathrm{CDS}}(\boldsymbol{s},\boldsymbol{a};j\rightarrow i)\hat{Q}(\boldsymbol{s},\boldsymbol{a},i) \right] \right. \\ & \left. - \mathbb{E}_{\boldsymbol{s},\boldsymbol{a}\sim\mathcal{D}_j} \left[w_{\mathrm{CDS}}(\boldsymbol{s},\boldsymbol{a};j\rightarrow i)\hat{Q}(\boldsymbol{s},\boldsymbol{a},i) \right] \right] \Big) \\ & + \frac{1}{2}\mathbb{E}_{j\sim[N],(\boldsymbol{s},\boldsymbol{a},\boldsymbol{s}')\sim\mathcal{D}_j} \left[w_{\mathrm{CDS}}(\boldsymbol{s},\boldsymbol{a};j\rightarrow i) \left(\hat{Q}(\boldsymbol{s},\boldsymbol{a},i) - \left(r(\boldsymbol{s},\boldsymbol{a},i)1\{\}\{j=i\} + \gamma Q(\boldsymbol{s}',\boldsymbol{a}') \right) \right)^2 \right] \Big],\end{aligned}$$

and

$$\pi \leftarrow \arg\max_{\pi'} \mathbb{E}_{i\sim[N]}\left[\mathbb{E}_{j\sim[N],\boldsymbol{s}\sim\mathcal{D}_j,\boldsymbol{a}\sim\pi'(\cdot|\boldsymbol{s},i)}\left[w_{\text{Cal-QL}}(\boldsymbol{s},\boldsymbol{a};j\rightarrow i)\hat{Q}^{\pi}(\boldsymbol{s},\boldsymbol{a},i)\right]\right],$$

where we use $k = 90$ in the multi-task Meta-World domain and $k = 50$ in the other multi-task domains.

To compute the weight w_{CDS}, we pick τ, i.e. the temperature term, using the exponential running average of $\Delta(\boldsymbol{s},\boldsymbol{a};\text{U}\rightarrow\text{L})$ in the single-task setting or $\Delta(\boldsymbol{s},\boldsymbol{a};j\rightarrow i)$ in the multi-task setting with decay 0.995 following [366]. Following [366] again, we clip the automatically chosen τ with a minimum and maximum threshold, which we directly use the values from [366]. We use $[1,\infty]$ as the minimum and maximum threshold for all state-based single-task and multi-task domains whereas the vision-based robotic manipulation domain does not require such clipping.

In the single-task experiments, we use a total batch size of 256 and balance the number of transitions sampled from $\mathcal{D}_{\text{L}}$ and $\mathcal{D}_{\text{U}}$ in each batch. In the multi-task experiments, following the training protocol in [366], for experiments with low-dimensional inputs, we use a stratified batch with 128 transitions for each task to train the Q-functions and the policy. We also balance the numbers of transitions sampled from the original task and the number of transitions drawn from other task data. Specifically, for each task i, we sample 64 transitions from $\mathcal{D}_i$ and the remaining 64 transitions from $\cup_{j\neq i}\mathcal{D}_{j\rightarrow i}$. In CDS+UDS, for each task $i\in[N]$, we only apply $w_{\text{CDS+UDS}}$ to data shared from other tasks on multi-task Meta-World environments and multi-task vision-based robotic manipulation tasks while we also apply the relabeling weight to transitions sampled from the original task dataset $\mathcal{D}_i$ with 50% probability in the multi-task AntMaze domain.

Regarding the choices of the architectures, for state-based domains, we use 3-layer feedforward neural networks with 256 hidden units for both the Q-networks and the policy. In the multi-task domains, we condition the policy and the Q-functions on a one-hot task ID, which is appended to the input state. In domains with high-dimensional image inputs, we adopt the multi-headed convolutional neural networks used in [162, 366]. We use images with dimension $472\times472\times3$, extra state features $(g_{\text{robot\_status}}, g_{\text{height}})$ and the one-hot task vector as the observations similar [162, 366]. Following the set-up in [161, 162, 366], we use Cartesian space control of the end-effector of the robot in 4D space (3D position and azimuth angle) along with two binary actions to open/close the gripper and terminate the episode respectively to represent the actions. For more details, see [161, 162].

E.11.2 *Experimental Details*

In this subsection, we include the discussion of the details the environment and datasets used for evaluating UDS and CDS+UDS. Note that all of our single-task datasets and environments are from the standard benchmark D4RL [92] while all of our multi-task

environment and offline datasets are from prior work [366]. We will nonetheless discuss the details to make our work self-contained. We acknowledge that all datasets with low-dimensional inputs are under the MIT License.

SINGLE-TASK HOPPER DOMAINS. We use the hopper environment and datasets from D4RL [92]. We consider the following three datasets, hopper-random, hopper-medium and hopper-expert. We construct the seven combinations of different data compositions using the three datasets as discussed in Table 17. To construct the combination, we take the first 10k transitions from labeled dataset and concatenate these labeled transitions with the entire unlabeled dataset with 1M transitions.

SINGLE-TASK ANTMAZE DOMAINS. We use the AntMaze environment and datasets from D4RL [92] where we consider two datasets, antmaze-medium-play and antmaze-large-play. These two datasets only contain 1M sub-optimal transitions that navigates to random or fixed locations that are different from the target task during evaluation. We use these two datasets as unlabeled datasets. For the labeled dataset, we use 10 expert demonstrations of solving the target task used in prior work [355].

MULTI-TASK META-WORLD DOMAINS. We use the door open, door close, drawer open and drawer close environments introduced in [366] from the public Meta-World [364] repo[2]. In this multi-task Meta-World environment, a door and a drawer are put on the same scene, which ensures that all four tasks share the same state space. The environment uses binary rewards for each task, which are adapted from the success condition defined in the Meta-World public repo. In this case, the robot gets a reward of 1 if it solves the target task and 0 otherwise. We use a fixed 200 timesteps for each episode and do not terminate the episode when receiving a reward of 1 at an intermediate timestep. We use large datasets with wide coverage of the state space and 152K transitions for the door open and drawer close tasks and datasets with limited (2K transitions), but optimal demonstrations for the door close and drawer open tasks.

We direct use the offline datasets constructed in [366], which are generated by training online SAC policies for each task with the dense reward defined in the Meta-World repo for 500 epochs. The medium-replay datasets use the whole replay buffer of the online SAC agent until 150 epochs while the expert datasets are collected by the final online SAC policy.

MULTI-TASK ANTMAZE DOMAINS. Following [366], we use the antmaze-medium-play and antmaze-large-play datasets from D4RL [92] and partitioning the datasets into multi-task datasets in an undirected way defined in [366]. Specifically, the dataset is randomly splitted into chunks with equal size, and then each chunk is assigned to a randomly

[2] The Meta-World environment can be found at the open-sourced repo https://github.com/rlworkgroup/metaworld

chosen task. Therefore, under such a task construction scheme, the task data for each task is of low success rate for the particular task it is assigned to and it is imperative for the multi-task offline RL algorithm to leverage effective data sharing strategy to achieve good performance. In AntMaze, we also use a binary reward, which provides the agent a reward of +1 when the ant reaches a position within a 0.5 radius of the task goal, which is also the reward used default by Fu et al. [92]. The terminal of an episode is set to be true when a reward of +1 is observed. We terminate the episode upon seeing a reward of 1 with the maximum possible 1000 transitions per episode. Following [366], we modify the datasets introduced by Fu et al. [92] by equally dividing the large dataset into different parts for different tasks, where each task corresponds to a different goal position.

MULTI-TASK WALKER DOMAINS. We have presented the details of the multi-task walker environment in Appendix E.8.

MULTI-TASK IMAGE-BASED ROBOTIC PICKING AND PLACING DOMAINS. Following [162, 366], we use sparse rewards for each task. That is, reward 1 is assigned to episodes that meet the success conditions and 0 otherwise. The success conditions are defined in [162]. We directly use the dataset used in [366]. Such a dataset is collected by first training a policy for each individual task using QT-Opt [161] until the success rate reaches 40% and 80% for picking tasks and placing tasks respectively and then combine the replay buffers of all tasks as the multi-task offline dataset. Note that the success rate of placing is higher because the robot is already holding the object at the start of the placing tasks, making the placing easier to solve. The dataset consists of a total number of 100K episodes with 25 transitions for each episode.

F

APPENDIX: ONLINE RL FINE-TUNING OF OFFLINE RL

F.1 IMPLEMENTATION DETAILS OF CAL-QL

Our algorithm, Cal-QL is illustrated in Algorithm 7.

F.1.1 *Cal-QL Algorithm*

We use $J_Q(\theta)$ to denote the conservative regularizer for the Q network update:

$$J_Q(\theta) := \alpha \underbrace{\left(\mathbb{E}_{\boldsymbol{s}\sim\mathcal{D},\boldsymbol{a}\sim\pi}\left[\max\left(Q_\theta(s,a), Q^\mu(s,a)\right)\right] - \mathbb{E}_{s,a\sim\mathcal{D}}\left[Q_\theta(s,a)\right]\right)}_{\text{Calibrated conservative regularizer } \mathcal{R}(\theta)} \tag{F.1.1}$$

$$+ \frac{1}{2}\mathbb{E}_{s,a,s'\sim\mathcal{D}}\left[\left(Q_\theta(s,a) - \mathcal{B}^\pi \boldsymbol{a}rQ(s,a)\right)^2\right]. \tag{F.1.2}$$

Algorithm 7 Cal-QL pseudo-code

1: Initialize Q-function, Q_θ, a policy, π_ϕ
2: **for** step t in $\{1, \ldots, N\}$ **do**
3: Train the Q-function using the conservative regularizer in Eq. F.1.1:

$$\theta_t := \theta_{t-1} - \eta_Q \nabla_\theta J_Q(\theta) \tag{F.1.3}$$

4: Improve policy π_ϕ with SAC-style update:

$$\phi_t := \phi_{t-1} + \eta_\pi \mathbb{E}_{\boldsymbol{s}\sim\mathcal{D},\boldsymbol{a}\sim\pi_\phi(\cdot|\boldsymbol{s})}\left[Q_\theta(\boldsymbol{s},\boldsymbol{a}) - \log \pi_\phi(\boldsymbol{a}|\boldsymbol{s})\right] \tag{F.1.4}$$

5: **end for**

F.2 REGRET ANALYSIS OF CAL-QL

We provide a theoretical version of Cal-QL in Algorithm 8. Policy fine-tuning has been studied in different settings [351, 308, 334]. Our analysis largely adopts the settings and results in Song et al. [308], with additional changes in Assumption F.2.1, Assumption F.2.3, and Definition F.2.4. Note that the goal of this proof is to demonstrate that *a pessimistic functional class (Assumption F.2.1)* allows one to utilize the offline data efficiently, rather than providing a new analytical technique for regret analysis. Note that we use f instead of Q_θ in the main text to denote the estimated Q function for notation simplicity.

Algorithm 8 Theoretical version of Cal-QL

Input: Value function class $\mathcal{F}$, # total iterations K, offline dataset $\mathcal{D}^\nu_h$ of size m_{off} for $h \in [H-1]$.
Initialize $f^1_h(s,a) = 0, \forall (s,a)$.
for $k = 1, \ldots, K$ **do**
  Let π^t be the greedy policy w.r.t. f^k ▷ I.e., $\pi^k_h(s) = \arg\max_a f^k_h(s,a)$.
  For each h, collect m_{on} online tuples $\mathcal{D}^k_h \sim d^{\pi^k}_h$ ▷ online data collection
  Set $f^{k+1}_H(s,a) = 0, \forall (s,a)$.
  for $h = H-1, \ldots 0$ **do** ▷ FQI with offline and online data
    Estimate f^{k+1}_h using conservative least squares on the aggregated data: ▷ I.e., CQL regularized class $\mathcal{C}_h$

$$f^{k+1}_h \leftarrow \arg\min_{f \in \mathcal{C}_h} \left\{ \widehat{\mathbb{E}}_{\mathcal{D}^\nu_h} \left[f(s,a) - r - \max_{a'} f^{k+1}_{h+1}(s',a') \right]^2 + \sum_{\tau=1}^{K} \widehat{\mathbb{E}}_{\mathcal{D}^\tau_h} \left[f(s,a) - r - \max_{a'} f^{k+1}_{h+1}(s',a') \right]^2 \right\} \tag{F.2.1}$$

    $f^{k+1}_h = \max\{f^{k+1}_h, Q^{\text{ref}}_h\}$ ▷ Set a reference policy for calibration (Definition 8.4.1)
  end for
end for
Output: π^K

F.2.1 *Notations*

- Feature covariance matrix $\Sigma_{k;h}$:

$$\boldsymbol{\Sigma}_{k;h} = \sum_{\tau=1}^{k} X_h(f^\tau)(X_h(f^\tau))^\top + \lambda \boldsymbol{I} \tag{F.2.2}$$

- Matrix Norm Zanette et al. [370]: for a matrix Σ, the matrix norm $\|\boldsymbol{u}\| \boldsymbol{\Sigma}$ is defined as:

$$\|\boldsymbol{u}\| \boldsymbol{\Sigma} = \sqrt{\boldsymbol{u}\boldsymbol{\Sigma}\boldsymbol{u}^\top} \tag{F.2.3}$$

- Weighted ℓ^2 norm: for a given distribution $\beta \in \Delta(\mathcal{S} \times \mathcal{A})$ and a function $f : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$, we denote the weighted ℓ^2 norm as:

$$\|f\|_{2,\beta^2} := \sqrt{\mathbb{E}_{(s,a)\sim\beta} f^2(s,a)} \tag{F.2.4}$$

- A stochastic reward function $R(s,a) \in \Delta([0,1])$
- For each offline data distribution $\nu = \{\nu_0, \ldots, \nu_{H-1}\}$, the offline data set at time step h (ν_h) contains data samples (s,a,r,s'), where $(s,a) \sim \nu_h, r \in R(s,a), s' \sim P(s,a)$.
- Given a policy $\pi := \{\pi_0, \ldots, \pi_{H-1}\}$, where $\pi_h : \mathcal{S} \mapsto \Delta(\mathcal{A})$, $d_h^\pi \in \Delta(s,a)$ denotes the state-action occupancy induced by π at step h.
- We consider the value-based function approximation setting, where we are given a function class $\mathcal{C} = \mathcal{C}_0 \times \ldots \mathcal{C}_{H-1}$ with $\mathcal{C}_h \subset \mathcal{S} \times \mathcal{A} \mapsto [0, V_{\max}]$.
- A policy π^f is defined as the greedy policy w.r.t. f: $\pi_h^f(s) = \arg\max_a f_h(s,a)$. Specifically, at iteration k, we use π^k to denote the greedy policy w.r.t. f^k.

F.2.2 *Assumptions and Defintions*

Assumption F.2.1 (Pessimistic Realizability and Completeness). *For any policy π^e, we say $\mathcal{C}_h$ is a pessimistic function class w.r.t. π^e, if for any h, we have $Q_h^{\pi^e} \in \mathcal{C}_h$, and additionally, for any $f_{h+1} \in \mathcal{C}_{h+1}$, we have $\mathcal{T} f_{h+1} \in \mathcal{C}_h$ and $f_h(s,a) \leq Q_h^{\pi^e}(s,a), \forall (s,a) \in \mathcal{S} \times \mathcal{A}$.*

Definition F.2.2 (Bilinear model Du et al. [66]). *We say that the MDP together with the function class $\mathcal{F}$ is a bilinear model of rand d of for any $h \in [H-1]$, there exist two (known) mappings $X_h, W_h : \mathcal{F} \mapsto \mathbb{R}^d$ with $\max_f \|X_h(f)\|_2 \leq B_X$ and $\max_f \|W_h(f)\|_2 \leq B_W$ such that*

$$\forall f, g \in \mathcal{F} : \quad \left| \mathbb{E}_{s,a\sim d_h^{\pi^f}} [g_h(s,a) - \mathcal{T} g_{h+1}(s,a)] \right| = |\langle X_h(f), W_h(g) \rangle| . \tag{F.2.5}$$

Assumption F.2.3 (Bilinear Rank of Reference Policies). *Suppose $Q^{\text{ref}} \in \mathcal{C}_{\text{ref}} \subset \mathcal{C}$, where $\mathcal{C}_{\text{ref}}$ is the function class of our reference policy, we assume the Bilinear rank of $\mathcal{C}_{\text{ref}}$ is d_{ref} and $d_{\text{ref}} \leq d$.*

Definition F.2.4 (Calibrated Bellman error transfer coefficient). *For any policy π, we define the calibrated transfer coefficient w.r.t. to a reference policy π^{ref} as*

$$C_\pi^{\text{ref}} := \max_{f \in \mathcal{C}, f(s,a) \geq Q^{\text{ref}}(s,a)} \frac{\sum_{h=0}^{H-1} \mathbb{E}_{s,a\sim d_h^\pi} [\mathcal{T} f_{h+1}(s,a) - f_h(s,a)]}{\sqrt{\sum_{h=0}^{H-1} \mathbb{E}_{s,a\sim \nu_h} (\mathcal{T} f_{h+1}(s,a) - f_h(s,a))^2}}, \tag{F.2.6}$$

where $Q^{\text{ref}} = Q^{\pi^{\text{ref}}}$.

F.2.3 *Discussions on the Assumptions*

The pessimistic realizability and completeness assumption (Assumption F.2.1) is motivated by some theoretical studies of the pessimistic offline methods [350, 47] with regularizers:

$$\min_{\theta} \alpha \underbrace{\left(\mathbb{E}_{s\sim\mathcal{D},a\sim\pi}\left[Q_\theta(s,a)\right] - \mathbb{E}_{s,a\sim\mathcal{D}}\left[Q_\theta(s,a)\right]\right)}_{\text{Conservative regularizer } \mathcal{R}(\theta)} + \frac{1}{2}\mathbb{E}_{s,a,s'\sim\mathcal{D}}\left[\left(Q_\theta(s,a) - \mathcal{B}^\pi \boldsymbol{a}rQ(s,a)\right)^2\right]. \tag{F.2.7}$$

Since the goal of the conservative regularizer $\mathcal{R}(\theta)$ intrinsically wants to enforce

$$Q_\theta(s,\pi(s)) \leq Q_\theta(s,\pi^e(s)), \tag{F.2.8}$$

where π is the training policy and π^e is the reference (behavior) policy. One can consider equation F.2.7 as the Lagrange duality formulation of the following primal optimization problem:

$$\min_{\theta} \mathbb{E}_{s,a,s'\sim\mathcal{D}}\left[\left(Q_\theta(s,a) - \mathcal{B}^\pi \boldsymbol{a}rQ(s,a)\right)^2\right], \text{ subject to } \mathbb{E}_{s\sim\mathcal{D},a\sim\pi}\left[Q_\theta(s,a)\right] \leq \mathbb{E}_{s\sim\mathcal{D},a\sim\pi^e}\left[Q_\theta(s,a)\right], \tag{F.2.9}$$

where the constraint set is equivalent to Assumption F.2.1. Although Assumption F.2.1 directly characterizes the constraint set of the primal form of equation F.2.7 the exact theoretical connection between the pessimistic realizability and the regularized bellman consistency equation is beyond the scope of this work and we would like to leave that for future studies.

Assumption F.2.1 allows us to restrict the functional class of interest to a smaller conservative function class $\mathcal{C} \subset \mathcal{F}$, which leads to a smaller Bellman rank of the reference policy ($d_{\text{ref}} \leq d$) suggested in Assumption F.2.3, and a smaller concentrability coefficient ($C_\pi^{\text{ref}} \leq C_\pi$) defined in Definition F.2.4, and F.3.2. Assumption F.2.3 and Definition F.3.2 provide the Bellman Bilinear rank and Bellman error transfer coefficient of the pessimistic functional class $\mathcal{C}$ of interest.

F.2.4 *Proof Structure Overview*

We provide an overview of the proof structure and its dependency on different assumptions below:

- Theorem F.2.5: the total regret is decomposed into *offline regrets* and *online regrets*.
  - Bounding *offline regrets*, requiring Definition F.2.4 and the following lemmas:
    - Performance difference lemma w.r.t. a comparator policy (Lemma F.3.5).

* Least square generalization bound (Lemma F.3.4), requiring Assumption F.2.1.

– Bounding *online regrets*, requiring Definition F.2.2

* Performance difference lemma for the online error (Lemma F.3.6).
* Least square generalization bound (Lemma F.3.4), requiring Assumption F.2.1.
* Upper bounds with the bilinear model assumption (Lemma F.3.7).
* Applying Elliptical Potential Lemma [195] with bellman rank d and d_{ref} (Lemma F.3.8), requiring Assumption F.2.3.

F.2.5 *Our Results*

Theorem F.2.5 (Formal Result of Theorem 8.6.1)**.** *Fix $\delta \in (0,1), m_{\text{off}} = K, m_{\text{on}} = 1$, suppose and the function class $\mathcal{C}$ follows Assumption F.2.1 w.r.t. π^e. Suppose the underlying MDP admits Bilinear rank d on function class $\mathcal{C}$ and d_{ref} on $\mathcal{C}_{\text{ref}}$, respectively, then with probability at least $1-\delta$, Algorithm 8 obtains the following bound on cumulative suboptimality w.r.t. any comparator policy π^e:*

$$\sum_{t=1}^{K} V^{\pi^e} - V^{\pi^k} = \widetilde{O}\left(\min\left\{C_{\pi^e}^{\text{ref}} H\sqrt{dK\log\left(|\mathcal{F}|/\delta\right)},\ K\left(V^{\pi^e} - V^{\text{ref}}\right) + H\sqrt{d_{\text{ref}} K\log\left(|\mathcal{F}|/\delta\right)}\right\}\right). \tag{F.2.10}$$

Note that Theorem F.2.5 provides a guarantee for *any* comparator policy π^{e}, which can be directly applied to $\pi^\star$ described in our informal result (Theorem 8.6.1). We also change the notation for the reference policy from μ in Theorem 8.6.1 to π^{ref} (similarly, $d_{\text{ref}}, V^{\text{ref}}, C_{\pi^e}^{\text{ref}}$ correspond to $d_\mu, V^\mu, C_{\pi^e}^{\mu}$ in Theorem 8.6.1) for notation consistency in the proof. Our proof of Theorem F.2.5 largely follows the proof of Theorem 1 of [308], and the major changes are caused by Assumption F.2.1, Assumption F.2.3, and Definition F.2.4.

Proof. Let $\mathcal{E}_k$ denote the event that $\left\{f_0^k(s,a) \le Q^{\text{ref}}(s,a)\right\}$ and $\boldsymbol{ar}\mathcal{E}_k$ denote the event that $\left\{f_0^k(s,a) > Q^{\text{ref}}(s,a)\right\}$. Let $V^{\text{ref}}(s) = \max_a Q^{\text{ref}}(s,a)$, we start by noting that

$$
\begin{aligned}
&\sum_{k=1}^{K} V^{\pi^e} - V^{\pi^{f^k}} = \sum_{k=1}^{K} \mathbb{E}_{s\sim\rho}\left[V_0^{\pi^e}(s) - V_0^{\pi^{f^k}}(s)\right] \\
&= \underbrace{\sum_{k=1}^{K} \mathbb{E}_{s\sim\rho}\left[1\left\{\boldsymbol{ar}\mathcal{E}_k\right\}\left(V_0^{\pi^e}(s) - V^{\text{ref}}(s)\right)\right]}_{\Gamma_0} + \underbrace{\sum_{k=1}^{K} \mathbb{E}_{s\sim\rho}\left[1\left\{\boldsymbol{ar}\mathcal{E}_k\right\}\left(V^{\text{ref}}(s) - \max_a f_0^k(s,a)\right)\right]}_{=0, \text{ by the definition of } \boldsymbol{ar}\mathcal{E}_k \text{ and line 9 of Algorithm 8}} \\
&\quad + \underbrace{\sum_{t=1}^{K} \mathbb{E}_{s\sim\rho}\left[1\left\{\boldsymbol{ar}\mathcal{E}_k\right\}\left(\max_a f_0^k(s,a) - V_0^{\pi^{f^k}}(s)\right)\right]}_{\Gamma_1} \\
&\quad + \underbrace{\sum_{k=1}^{K} \mathbb{E}_{s\sim\rho}\left[1\left\{\mathcal{E}_k\right\}\left(V_0^{\pi^e}(s) - \max_a f_0^k(s,a)\right)\right]}_{\Gamma_2} \\
&\quad + \underbrace{\sum_{t=1}^{T} \mathbb{E}_{s\sim\rho}\left[1\left\{\mathcal{E}_k\right\}\left(\max_a f_0^k(s,a) - V_0^{\pi^{f^k}}(s)\right)\right]}_{\Gamma_3}.
\end{aligned}
\tag{F.2.11}
$$

Let $K_1 = \sum_{k=1}^{K} 1\left\{f_0^k(s,a) > Q^{\text{ref}}(s,a)\right\}$ and $K_2 = \sum_{k=1}^{K} 1\left\{f_0^k(s,a) \le Q^{\text{ref}}(s,a)\right\}$ (or equivalently $K_1 = \sum_{k=1}^{K} 1\left\{\boldsymbol{ar}\mathcal{E}_k\right\}$, $K_2 = \sum_{k=1}^{K} 1\left\{\mathcal{E}_k\right\}$). For Γ_0, we have

$$
\Gamma_0 = K_2 \mathbb{E}_{s\sim\rho}\left(V^{\pi^e}(s) - V^{\text{ref}}(s)\right). \tag{F.2.12}
$$

For Γ_2, we have

$$
\begin{aligned}
\Gamma_2 &= \sum_{k=1}^{K} \mathbb{E}_{s\sim\rho}\left[1\left\{\mathcal{E}_k\right\}\left(V_0^{\pi^e}(s) - \max_a f_0^k(s,a)\right)\right] \\
&\overset{(i)}{\le} \sum_{k=1}^{K} 1\left\{\mathcal{E}_k\right\} \sum_{h=0}^{H-1} \mathbb{E}_{s,a\sim d_h^{\pi^e}}\left[\mathcal{T} f_{h+1}^k(s,a) - f_h^k(s,a)\right] \\
&\overset{(ii)}{\le} \sum_{k=1}^{K} \left[C_{\pi^e}^{\text{ref}} \cdot 1\left\{\mathcal{E}_k\right\} \sqrt{\sum_{h=0}^{H-1} \mathbb{E}_{s,a\sim\nu_h}\left[\left(f_h^k(s,a) - \mathcal{T} f_{h+1}^k(s,a)\right)^2\right]}\right] \\
&\overset{(iii)}{\le} K_1 C_{\pi^e}^{\text{ref}} \sqrt{H \cdot \Delta_{\text{off}}},
\end{aligned}
\tag{F.2.13}
$$

where Δ_{off} is similarly defined as Song et al. [308] (See equation F.3.3 of Lemma F.3.4). Inequality (i) holds because of Lemma F.3.5, inequality (ii) holds by the definition of $C_{\pi^e}^{\text{ref}}$ (Definition F.2.4), inequality (iii) holds by applying Lemma F.3.4 with the function class satisfying Assumption F.2.1, and Definition F.2.4. Note that the telescoping decomposition technique in the above equation also appears in [349, 151, 66]. Next, we will bound $\Gamma_1 + \Gamma_3$:

$$\begin{aligned}
\Gamma_1 + \Gamma_3 &= \sum_{k=1}^{K} \left(1\left\{\mathcal{E}_k\right\} + 1\left\{\boldsymbol{ar}\mathcal{E}_k\right\}\right) \mathbb{E}_{s\sim d_0}\left[\max_a f_0^k(s,a) - V_0^{\pi^{f^k}}(s)\right] \\
&\overset{(i)}{\leq} \sum_{k=1}^{K} \left(1\left\{\mathcal{E}_k\right\} + 1\left\{\boldsymbol{ar}\mathcal{E}_k\right\}\right) \sum_{h=0}^{H-1} \left|\mathbb{E}_{s,a\sim d_h^{\pi^{f^k}}}\left[f_h^k(s,a) - \mathcal{T} f_{h+1}^k(s,a)\right]\right| \\
&\overset{(ii)}{=} \sum_{t=1}^{K} \left[\left(1\left\{\mathcal{E}_k\right\} + 1\left\{\boldsymbol{ar}\mathcal{E}_k\right\}\right) \sum_{h=0}^{H-1} \left|\left\langle X_h(f^k), W_h(f^k)\right\rangle\right|\right] \\
&\overset{(iii)}{\leq} \sum_{k=1}^{K} \left[\left(1\left\{\mathcal{E}_k\right\} + 1\left\{\boldsymbol{ar}\mathcal{E}_k\right\}\right) \sum_{h=0}^{H-1} \left\|X_h(f^k)\right\| \boldsymbol{\Sigma}_{k-1;h}^{-1} \sqrt{\Delta_{\text{on}} + \lambda B_W^2}\right],
\end{aligned} \tag{F.2.14}$$

where Δ_{on} is similarly defined as Song et al. [308] (See equation F.3.4 of Lemma F.3.4). Inequality (i) holds by Lemma F.3.6, equation (ii) holds by the definition of Bilinear model (equation F.2.5 in Definition F.2.2), inequality (iii) holds by Lemma F.3.7 and Lemma F.3.4 with the function class satisfying Assumption F.2.1. Using Lemma F.3.8, we have that

$$\begin{aligned}
&\Gamma_1 + \Gamma_3 \\
&\leq \sum_{k=1}^{K} \left[\left(1\left\{\mathcal{E}_k\right\} + 1\left\{\boldsymbol{ar}\mathcal{E}_k\right\}\right) \sum_{h=0}^{H-1} \left\|X_h(f^k)\right\| \boldsymbol{\Sigma}_{k-1;h}^{-1} \sqrt{\Delta_{\text{on}} + \lambda B_W^2}\right] \\
&\overset{(i)}{\leq} H\sqrt{2d \log\left(1 + \frac{K_1 B_X^2}{\lambda d}\right) \cdot (\Delta_{\text{on}} + \lambda B_W^2) \cdot K_1} + H\sqrt{2d_{\text{ref}} \log\left(1 + \frac{K_2 B_X^2}{\lambda d_{\text{ref}}}\right) \cdot (\Delta_{\text{on}} + \lambda B_W^2) \cdot K_2} \\
&\overset{(ii)}{\leq} H\left(\sqrt{2d \log\left(1 + \frac{K_1}{d}\right) \cdot (\Delta_{\text{on}} + B_X^2 B_W^2) \cdot K_1} + \sqrt{2d_{\text{ref}} \log\left(1 + \frac{K_2}{d_{\text{ref}}}\right) \cdot (\Delta_{\text{on}} + B_X^2 B_W^2) \cdot K_2}\right),
\end{aligned} \tag{F.2.15}$$

where the first part of inequality (i) holds by the assumption that the underlying MDPs have bellman rank d (Definition F.2.2) when $\boldsymbol{ar}\mathcal{E}_k$ happens, and the second part of inequality (i) holds by the assumption that $\mathcal{C}_{\text{ref}}$ has bilinear rank d_{ref} (Assumption F.2.3) $\mathcal{C}_{\text{ref}}$ has bellman rank d_{ref} when $\mathcal{E}_k$ happens. Inequality (ii) holds by plugging in $\lambda =$

B_X^2. Substituting equation F.2.12, inequality F.2.13, and inequality equation F.2.15 into equation F.2.11, we have

$$\sum_{t=1}^{K} V^{\pi^e} - V^{\pi^{f^k}} \leq \Gamma_0 + \Gamma_2 + \Gamma_1 + \Gamma_3 \leq K_2 \left(V^{\pi^e}(s) - V^{\text{ref}}(s) \right) + K_1 C_{\pi^e}^{\text{ref}} \sqrt{H \cdot \Delta_{\text{off}}}$$
$$+ H \left(\sqrt{2d \log\left(1 + \frac{K_1}{d}\right) \cdot (\Delta_{\text{on}} + B_X^2 B_W^2) \cdot K_1} + \sqrt{2d_{\text{ref}} \log\left(1 + \frac{K_2}{d_{\text{ref}}}\right) \cdot (\Delta_{\text{on}} + B_X^2 B_W^2) \cdot K_2} \right) \tag{F.2.16}$$

Plugging in the values of $\Delta_{\text{on}}, \Delta_{\text{off}}$ from equation F.3.3 and equation F.3.4, and using the subadditivity of the square root function, we have

$$\begin{aligned}
&\sum_{k=1}^{K} V^{\pi^e} - V^{\pi^{f^k}} \\
&\leq K_2 \left(V^{\pi^e}(s) - V^{\text{ref}}(s) \right) + 16 V_{\max} C_{\pi^e}^{\text{ref}} K_1 \sqrt{\frac{H}{m_{\text{off}}} \log\left(\frac{2HK_1|\mathcal{F}|}{\delta}\right)} \\
&\quad + \left(16 V_{\max} \sqrt{\frac{1}{m_{\text{on}}} \log\left(\frac{2HK_1|\mathcal{F}|}{\delta}\right)} + B_X B_W \right) \cdot H \sqrt{2dK_1 \log\left(1 + \frac{K_1}{d}\right)} \\
&\quad + \left(16 V_{\max} \sqrt{\frac{1}{m_{\text{on}}} \log\left(\frac{2HK_2|\mathcal{F}|}{\delta}\right)} + B_X B_W \right) \cdot H \sqrt{2d_{\text{ref}} K_2 \log\left(1 + \frac{K_2}{d_{\text{ref}}}\right)}.
\end{aligned} \tag{F.2.17}$$

Setting $m_{\text{off}} = K, m_{\text{on}} = 1$ in the above equation completes the proof, we have

$$\begin{aligned}
&\sum_{k=1}^{K} V^{\pi^e} - V^{\pi^k} \\
&\leq \widetilde{O}\left(C_{\pi^e}^{\text{ref}} \sqrt{HK_1 \log\left(|\mathcal{F}|/\delta\right)} \right) + \widetilde{O}\left(H \sqrt{dK_1 \log\left(|\mathcal{F}|/\delta\right)} \right) \\
&\quad + K_2 \left(V^{\pi^e}(s) - V^{\text{ref}}(s) \right) + \widetilde{O}\left(H \sqrt{d_{\text{ref}} K_2 \log\left(|\mathcal{F}|/\delta\right)} \right) \\
&\leq \begin{cases} \widetilde{O}\left(C_{\pi^e}^{\text{ref}} H \sqrt{dK_1 \log\left(|\mathcal{F}|/\delta\right)} \right) & \text{if } K_1 \gg K_2, \\ \widetilde{O}\left(K_2 \left(V^{\pi^e} - V^{\text{ref}} \right) + H \sqrt{d_{\text{ref}} K_2 \log\left(|\mathcal{F}|/\delta\right)} \right) & \text{otherwise.} \end{cases} \\
&\leq \widetilde{O}\left(\min\left\{ C_{\pi^e}^{\text{ref}} H \sqrt{dK \log\left(|\mathcal{F}|/\delta\right)},\ K \left(V^{\pi^e} - V^{\text{ref}} \right) + H \sqrt{d_{\text{ref}} K \log\left(|\mathcal{F}|/\delta\right)} \right\} \right),
\end{aligned} \tag{F.2.18}$$

where the last inequality holds because $K_1, K_2 \leq K$, which completes the proof. □

F.3 KEY RESULTS OF HYQ [308]

In this section, we restate the major theoretical results of Hy-Q [308].

F.3.1 *Assumptions*

Assumption F.3.1 (Realizability and Bellman completeness). *For any h, we have $Q_h^\star \in \mathcal{F}_h$, and additionally, for any $f_{h+1} \in \mathcal{F}_{h+1}$, we have $\mathcal{T} f_{h+1} \in \mathcal{F}_h$.*

Definition F.3.2 (Bellman error transfer coefficient). *For any policy π, we define the transfer coefficient as*

$$C_\pi := \max\left\{0, \max_{f \in \mathcal{F}} \frac{\sum_{h=0}^{H-1} \mathbb{E}_{s,a\sim d_h^\pi}[\mathcal{T} f_{h+1}(s,a) - f_h(s,a)]}{\sqrt{\sum_{h=0}^{H-1} \mathbb{E}_{s,a\sim \nu_h}(\mathcal{T} f_{h+1}(s,a) - f_h(s,a))^2}}\right\}. \tag{F.3.1}$$

F.3.2 *Main Theorem of Hy-Q*

Theorem F.3.3 (Theorem 1 of Song et al. [308]). *Fix $\delta \in (0,1), m_{\text{off}} = K, m_{\text{on}} = 1$, and suppose that the underlying MDP admits Bilinear rank d (Definition F.2.2), and the function class $\mathcal{F}$ satisfies Assumption F.3.1. Then with probability at least $1-\delta$, HyQ obtains the following bound on cumulative suboptimality w.r.t. any comparator policy π^e:*

$$\text{Reg}(K) = \widetilde{O}\left(\max\{C_{\pi^e}, 1\} V_{\max} B_X B_W \sqrt{dH^2 K \cdot \log(|\mathcal{F}|/\delta)}\right). \tag{F.3.2}$$

F.3.3 *Key Lemmas*

F.3.3.1 *Least Squares Generalization and Applications*

Lemma F.3.4 (Lemma 7 of Song et al. [308], Online and Offline Bellman Error Bound for FQI). *Let $\delta \in (0,1)$ and $\forall h \in [H-1], k \in [K]$, let f_h^{k+1} be the estimated value function for time step h computed via least square regression using samples in the dataset $\{\mathcal{D}_h^\nu, \mathcal{D}_h^1, \ldots, \mathcal{D}_h^T\}$ in equation F.2.1 in the iteration t of Algorithm 8. Then with probability at least $1-\delta$, for any $h \in [H-1]$ and $k \in [K]$, we have*

$$\left\| f_h^{k+1} - \mathcal{T} f_{h+1}^{k+1} \right\| 2, {\nu_h}^2 \le \frac{1}{m_{\text{off}}} 256 V_{\max}^2 \log(2HK|\mathcal{F}|/\delta) =: \Delta_{\text{off}} \tag{F.3.3}$$

and

$$\sum_{\tau=1}^{k} \left\| f_h^{k+1} - \mathcal{T} f_{h+1}^{k+1} \right\| 2, {\mu_h^\tau}^2 \le \frac{1}{m_{\text{on}}} 256 V_{\max}^2 \log(2HK|\mathcal{F}|/\delta) =: \Delta_{\text{on}}, \tag{F.3.4}$$

where ν_h denotes the offline data distribution at time h, and the distribution $\mu_h^\tau \in \Delta(s,a)$ is defined such that $s,a \sim d_h^{\pi^\tau}$.

F.3.3.2 *Bounding Offline Suboptimality via Performance Difference Lemma*

Lemma F.3.5 (Lemma 5 of Song et al. [308], performance difference lemma of w.r.t. π^e)**.** *Let $\pi^e = (\pi_0^e, \dots, \pi_{H-1}^e)$ be a comparator policy and consider any value function $f = (f_0, \dots, f_{H-1})$, where $f_h : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$. Then we have*

$$\mathbb{E}_{s\sim d_0}\left[V_0^{\pi^e}(s) - \max_a f_0(s,a)\right] \leq \sum_{i=1}^{H-1} \mathbb{E}_{s,a\sim d_i^{\pi^e}}\left[\mathcal{T} f_{i+1}(s,a) - f_i(s,a)\right], \tag{F.3.5}$$

where we define $f_H(s,a) = 0, \forall (s,a)$.

F.3.3.3 *Bounding Online Suboptimality via Performance Difference Lemma*

Lemma F.3.6 (Lemma 4 of Song et al. [308], performance difference lemma)**.** *For any function $f = (f_0, \dots, f_{H-1})$ where $f_h : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ and $h \in [H-1]$, we have*

$$\mathbb{E}_{s\sim d_0}\left[\max_a f_0(s,a) - V_0^{\pi^f}(s)\right] \leq \sum_{h=0}^{H-1} \left|\mathbb{E}_{s,a\sim d_h^{\pi^f}}\left[f_h(s,a) - \mathcal{T} f_{h+1}(s,a)\right]\right|, \tag{F.3.6}$$

where we define $f_H(s,a) = 0, \forall s, a$.

Lemma F.3.7 (Lemma 8 of Song et al. [308], upper bounding bilinear class)**.** *For any $k \geq 2$ and $h \in [H-1]$, we have*

$$\left|\left\langle W_h(f^k), X_h(f^k)\right\rangle\right| \leq \left\|X_h(f^k)\right\|_{\mathbf{\Sigma}_{k-1;h}^{-1}} \sqrt{\sum_{i=1}^{k-1} \mathbb{E}_{s,a\sim d_h^{f^i}}\left[\left(f_h^k - \mathcal{T} f_{h+1}^k\right)^2\right] + \lambda B_W^2}, \tag{F.3.7}$$

where $\mathbf{\Sigma}_{k-1;h}$ is defined as equation F.2.2 and we use $d_h^{f^k}$ to denote $d_h^{\pi^{f^k}}$.

Lemma F.3.8 (Lemma 6 of Song et al. [308], bounding the inverse covariance norm)**.** *Let $X_h(f^1), \dots, X_h(f^K) \in \mathbb{R}^d$ be a sequence of vectors with $\left\|X_h(f^k)\right\| 2 \leq B_X < \infty, \forall k \leq K$. Then we have*

$$\sum_{k=1}^{K} \left\|X_h(f^k)\right\|_{\mathbf{\Sigma}_{k-1;h}^{-1}} \leq \sqrt{2dK \log\left(1 + \frac{KB_X^2}{\lambda d}\right)}, \tag{F.3.8}$$

where we define $\mathbf{\Sigma}_{k;h} := \sum_{\tau=1}^{k} X_h(f^\tau) X_h(f^\tau)^T + \lambda \boldsymbol{I}$ and we assume $\lambda \geq B_X^2$ holds $\forall k \in [K]$.

F.4 ENVIRONMENT DETAILS

F.4.1 *Antmaze*

The Antmaze navigation tasks aim to control an 8-DoF ant quadruped robot to move from a starting point to a desired goal in a maze. The agent will receive sparse +1/0 rewards

depending on whether it reaches the goal or not. We study each method on "medium" and "hard" (shown in Figure 34) mazes which are difficult to solve, using the following datasets from D4RL [92]: large-diverse, large-play, medium-diverse, and medium-play. The difference between "diverse" and "play" datasets is the optimality of the trajectories they contain. The "diverse" datasets contain the trajectories commanded to a random goal from random starting points, while the "play" datasets contain the trajectories commanded to specific locations which are not necessarily the goal. We used an episode length of 1000 for each task. For Cal-QL, CQL, and IQL, we pre-trained the agent using the offline dataset for 1M steps. We then trained online fine-tuning for 1M environment steps for each method.

F.4.2 *Franka Kitchen*

The Franka Kitchen domain require controlling a 9-DoF Franka robot to arrange a kitchen environment into a desired configuration. The configuration is decomposed into 4 subtasks, and the agent will receive rewards of 0, $+1$, $+2$, $+3$, or $+4$ depending on how many subtasks it has managed to solve. To solve the whole task and reach the desired configuration, it is important to learn not only how to solve each subtask, but also to figure out the correct order to solve. We study this domain using datasets with three different optimalities: kitchen-complete, kitchen-partial, and kitchen-mixed. The "complete" dataset contains the trajectories of the robot performing the whole task completely. The "partial" dataset partially contains some complete demonstrations, while others are incomplete demonstrations solving the subtasks. The "mixed" dataset only contains incomplete data without any complete demonstrations, which is hard and requires the highest degree of stitching and generalization ability. We used an episode length of 1000 for each task. For Cal-QL, CQL, and IQL, we pre-trained the agent using the offline dataset for 500K steps. We then performed online fine-tuning for 1.25M environment steps for each method.

F.4.3 *Adroit*

The Adroit domain involves controlling a 24-DoF shadow hand robot. There are 3 tasks we consider in this domain: pen-binary, relocate-binary, relocate-binary. These tasks comprise a limited set of narrow human expert data distributions ($\sim$ 25) with additional trajectories collected by a behavior-cloned policy. We truncated each trajectory and used the positive segments (terminate when the positive reward signal is found) for all methods. This domain has a very narrow dataset distribution and a large action space. In addition, learning in this domain is difficult due to the sparse reward, which leads to exploration challenges. We utilized a variant of the dataset used in prior work [240] to have a standard comparison with SOTA offline fine-tuning experiments that consider this domain. For the offline learning phase, we pre-trained the agent for 20K steps. We then performed online fine-tuning for 300K environment steps for the pen-binary task, and 1M environment

steps for the door-binary and relocate-binary tasks. The episode length is 100, 200, and 200 for pen-binary, door-binary, and relocate-binary respectively.

F.4.4 *Visual Manipulation Domain*

The Visual Manipulation domain consists of a pick-and-place task. This task is a multitask formulation explored in the work, Pre-training for Robots (PTR) [187]. Here each task is defined as placing an object in a bin. A distractor object was present in the scene as an adversarial object which the agent had to avoid picking. There were 10 unique objects and no overlap between the task objects and the interfering/distractor objects. The episode length is 40. For the offline phase, we pre-trained the policy with offline data for 50K steps. We then performed online fine-tuning for 100K environment steps for each method, taking 5 gradient steps per environment step.

F.5 EXPERIMENT DETAILS

F.5.1 *Normalized Scores*

The visual-manipulation, adroit, and antmaze domains are all goal-oriented, sparse reward tasks. In these domains, we computed the normalized metric as simply the goal achieved rate for each method. For example, in the visual manipulation environment, if the object was placed successfully in the bin, a $+1$ reward was given to the agent and the task is completed. Similarly, for the door-binary task in the adroit tasks, we considered the success rate of opening the door. For the kitchen task, the task is to solve a series of 4 sub-tasks that need to be solved in an iterative manner. The normalized score is computed simply as $\frac{\text{\#tasks solved}}{\text{total tasks}}$.

F.5.2 *Mixing Ratio Hyperparameter*

In this work, we explore the mixing ratio parameter m, which is used during the online fine-tuning phase. The mixing ratio is either a value in the range $[0, 1]$ or the value -1. If this mixing ratio is within $[0, 1]$, it represents what percentage of offline and online data is seen in each batch when fine-tuning. For example, if the mixing ratio $m = 0.25$, that means for each batch we sample 25% from the offline data and 75% from online data. Instead, if the mixing ratio is -1, the buffers are appended to each other and sampled uniformly.

F.5.3 *Details and Hyperparameters for CQL and Cal-QL*

We list the hyperparameters for CQL and Cal-QL in Table 60. We utilized a variant of Bellman backup that computes the target value by performing a maximization over

target values computed for k actions sampled from the policy at the next state, where we used $k = 4$ in visual pick and place domain and $k = 10$ in others. In the Antmaze domain, we used the dual version of CQL [181] and conducted ablations over the value of the threshold of the CQL regularizer $\mathcal{R}(\theta)$ (target action gap) instead of α. In the visual-manipulation domain which is not presented in the original paper, we swept over the alpha values of $\alpha = 0.5, 1, 5, 10$, and utilized separate α values for offline ($\alpha = 5$) and online ($\alpha = 0.5$) phases for the final results. We built our code upon the CQL implementation from https://github.com/young-geng/JaxCQL [104]. We used a single NVIDIA TITAN RTX chip to run each of our experiments.

F.5.4 *Details and Hyperparameters for IQL*

We list the hyperparameters for IQL in Table 61. To conduct our experiments, we used the official implementation of IQL provided by the authors [175], and primarily followed their recommended parameters, which they previously ablated over in their work. In the visual-manipulation domain which is not presented in the original paper, we performed a parameter sweep over expectile $\tau = 0.5, 0.6, 0.7, 0.8, 0.9, 0.95, 0.99$ and temperature $\beta = 1, 3, 5, 10, 25, 50$ and selected the best-performing values of $\tau = 0.7$ and $\beta = 10$ for our final results. In addition, as the second best-performing method in the visual-manipulation domain, we also attempted to use separate β values for IQL, for a fair comparison with CQL and Cal-QL. However, we found that it has little to no effect, as shown in Figure 39.

F.5.5 *Details and Hyperparameters for AWAC and ODT*

We used the JAX implementation of AWAC from https://github.com/ikostrikov/jaxrl [173]. We primarily followed the author's recommended parameters, where we used the Lagrange multiplier $\lambda = 1.0$ for the Antmaze and Franka Kitchen domains, and $\lambda = 0.3$ for the Adroit domain. In the visual-manipulation domain, we performed a parameter sweep over $\lambda = 0.1, 0.3, 1, 3, 10$ and selected the best-performing value of $\lambda = 1$ for our final results. For ODT, we used the author's official implementation from https://github.com/facebookresearch/online-dt, with the author's recommended parameters they used in the Antmaze domain. In addition, in support of our result of AWAC and ODT (as shown in Table 20), the poor performance of Decision Transformers and AWAC in the Antmaze domain can also be observed in Table 1 and Table 2 of the IQL paper [175].

F.5.6 *Details and Hyperparameters for SAC, SAC + Offline Data, and CQL + SAC*

We used the standard hyperparameters for SAC as derived from the original implementation in [123]. We used the same other hyperparameters as CQL and Cal-QL. We used

automatic entropy tuning for the policy and critic entropy terms, with a target entropy of the negative action dimension. For SAC, the agent is only trained with the online explored data. For SAC + Offline Data, the offline data and online explored data is combined together and sampled uniformly. For Hybrid RL, we use the same mixing ratio used for CQL and Cal-QL presented in Table 60. For CQL + SAC, we first pre-train with CQL and then run online fine-tuning using both offline and online data, also using the same mixing ratio presented in Table 60.

Table 60: CQL, Cal-QL Hyperparameters

Hyperparameters	Adroit	Kitchen	Antmaze	Manipulation
α	1	5	-	5 (online: 0.5)
target action gap	-	-	0.8	-
mixing ratio	-1, 0.25, **0.5**	-1, **0.25**, 0.5	0.5	0.2, **0.5**, 0.7, 0.9

Table 61: IQL Hyperparameters

Hyperparameters	Adroit	Kitchen	Antmaze	Manipulation
expectile τ	0.8	0.7	0.9	0.7
temperature β	3	0.5	10	10
mixing ratio	-1, **0.2**, 0.5	-1, **0.25**, 0.5	0.5	0.2, **0.5**, 0.7, 0.9

F.6 DISCUSSION ON LIMITATIONS OF EXISTING FINE-TUNING METHODS

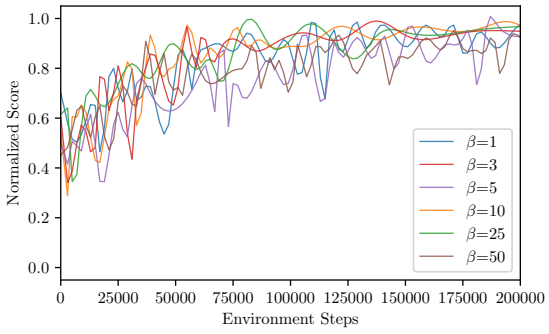


Figure 39: Abalation on IQL's online temperature values: The change in the temperature β used in online fine-tuning phase has little to no effect on the sample efficiency.

In this section, we aim to highlight some potential reasons behind the slow improvement of other methods in our empirical analysis experiment in Section 8.4.1, and specifically, we use IQL for the analysis. We first swept over the temperature β values used in the online fine-tuning phase for IQL, which controls the constraint on how closely the learned policy should match the behavior policy. As shown in Figure 39, the change in the temperature β has little to no effect on the sample efficiency. Another natural hypothesis is that IQL improves slowly because we are not making enough updates per unit of data

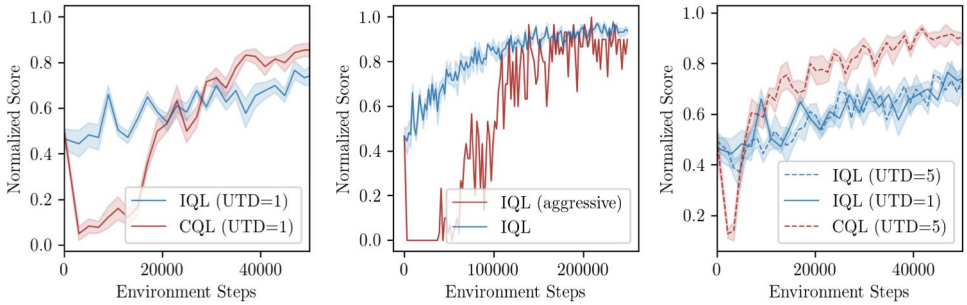


Figure 40: IQL and CQL: Step 0 on the x-axis is the performance after offline pre-training. Observe while CQL suffers from initial policy un[illegible] ghout fine-tuning.

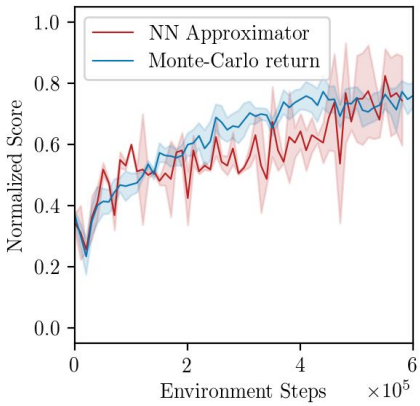


Figure 41: The performance of Cal-QL using a neural net approximator for the reference value function is comparable to using the Monte-Carlo return.

collected by the environment. To investigate this, we ran IQL with **(a)** five times as many gradient steps per step of data collection (UTD $= 5$), and **(b)** with a more aggressive policy update. Observe in Figure 40 that **(a)** does not improve the asymptotic performance of IQL, although it does improve CQL meaning that there is room for improvement on this task by making more gradient updates. Observe in Figure 40 that **(b)** often induces policy unlearning, similar to the failure mode in CQL. These two observations together indicate that a policy constraint approach can slow down learning asymptotically, and we cannot increase the speed by making more aggressive updates as this causes the policy to find erroneously optimistic out-of-distribution actions, and unlearn the policy learned from offline data.

F.7 Impact of estimation errors in the reference value function

In our experiments, we compute the reference value functions using Monte-Carlo return estimates. However, this may not be available in all tasks. How does Cal-QL behave when reference value functions must be estimated using the offline dataset itself? To answer this, we ran an experiment on the kitchen domain, where instead of using an estimate for Q^μ based on the Monte-Carlo return, we train a neural network function approximator

Q^{μ}_{θ} to approximate Q^{μ} via supervised regression on to Monte-Carlo return, which is then utilized by Cal-QL. Observe in Figure 41, that the performance of Cal-QL largely remains unaltered. This implies as long as we can obtain a reasonable function approximator to estimate the Q-function of the reference policy (in this case, the behavior policy), errors in the reference Q-function do not affect the performance of Cal-QL significantly.

F.8 INITIAL UNLEARNING OF CQL DURING ONLINE FINE-TUNING

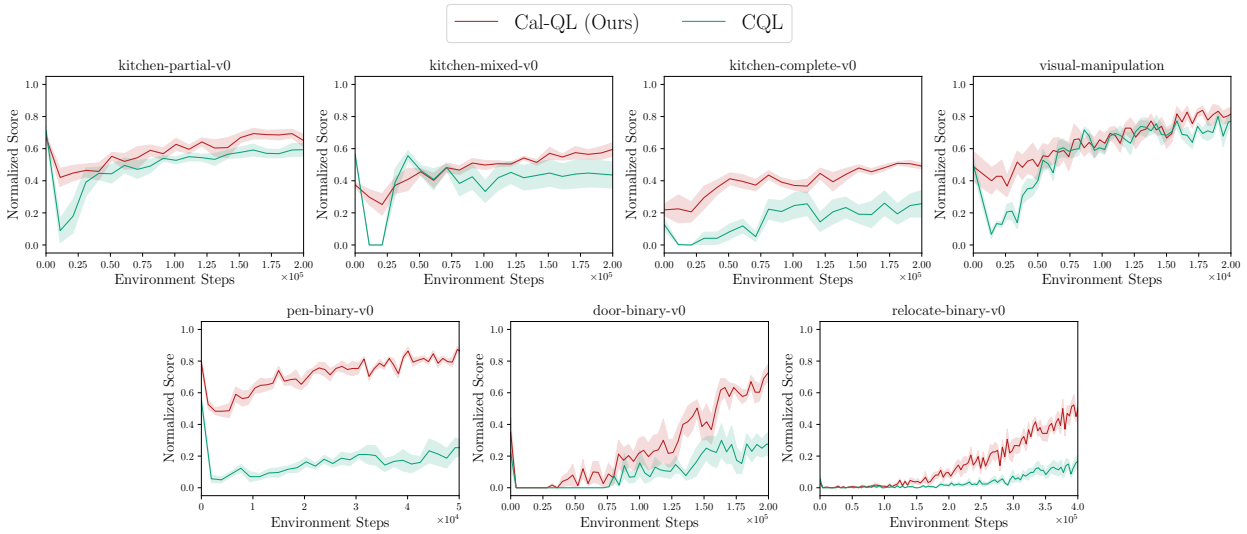


Figure 42: While CQL experiences initial unlearning, Cal-QL effectively mitigates it and quickly recovers its performance.

In this section, we show the learning curves of CQL and Cal-QL from Figure 35 and zoom in on the x-axis to provide a clearer visualization of CQL's initial unlearning in the Franka Kitchen, Adroit, and the visual-manipulation domains. As depicted in Figure 42, it is evident across all tasks that while CQL experiences initial unlearning, Cal-QL can effectively mitigate it and quickly recovers its performance. Regarding the Antmaze domain, as we discussed in section 8.7.3, CQL does not exhibit initial unlearning since the default dataset has a high coverage of data. However, we can observe a similar phenomenon if we narrow down the dataset distribution (as shown in Figure 37).

G

APPENDIX: REAL-ROBOT PRE-TRAINING

G.1 DIAGNOSTIC STUDY IN SIMULATION

We perform a diagnostic study in simulation to verify some of the insights observed in our real-world experiments. We created a bin sort task, where a WidowX250 robot is placed in front two bins and is provided with two objects (more details in Appendix G.2). The task is to sort each object in the correct bin associated with that object. The pre-training data provided to this robot is pick-place data, which only demonstrates how to pick *one* of the objects and place it in one of the bins, but does not demonstrate the compound task of placing both objects. In order to succeed at this such a compound task, a robot must learn an abstract representation of the skill of sorting an object during the pre-training phase and then figure out that it needs to apply this skill multiple times in a trajectory to succeed at the task from just *five* demonstrations of the desired sorting behavior.

The performance numbers (along with 95%-confidence intervals) are shown in Table 62. Observe that PTR improves upon prior methods in a statistically significant manner, outperforming the BC and COG baselines by a significant margin. This validates the efficacy of PTR in simulation and corroborates our real-world results.

G.2 DETAILS OF OUR EXPERIMENTAL SETUP

G.2.1 *Real-World Experimental Setup*

A picture of our real-world experimental setup is shown in Figure 43. The scenarios considered in our experiments (Section 9.4) are designed to evaluate the performance of our method under a variety of situations and therefore we set up these tasks in different toykitchen domains (see Figure 43) on three different WidowX 250 robot arms. We use data from the bridge dataset [70] consisting of data collected with many robots in many

Method	Success rate
BC (joint training)	7.00 ± 0.00 %
COG (joint training)	8.00 ± 1.00 %
BC (finetune)	4.88 ± 4.07 %
PTR (Ours)	**17.41 ± 1.77 %**

Table 62: Performance of PTR in comparison with other methods on the simulated bin sorting task, trained for many more gradient steps for all methods until each one of them converges. Observe that PTR substantially outperforms other prior methods, including joint training on the same data with BC or CQL. Training on target data only is unable to recover a non-zero performance, so we do not report it in this table. Since the 95%-confidence intervals do not overlap between PTR and other methods, it indicates that PTR improves upon baselines in a statistically significant manner.

domains for training but exclude the task/domain that we use for evaluation from the training dataset.

G.2.2 *Simulation Setup*

We evaluate our approach in a simulated bin-sorting task on the simulated WidowX 250 platform, aimed to mimic the setup we use for our real-world evaluations. This setup is designed in the PyBullet simulation framework provided by Singh et al. [303]. A picture is shown in Figure 44. In this task, two different bins and two different objects are placed in front of the WidowX robot. The goal of the robot is to correctly sort each of the two objects to their designated bin (e.g the cylinder is supposed to be placed in the left bin and the teapot should be placed in the right bin. We refer to this task as a *compound* task since it requires successfully combining behaviors of two different pick-and-place skills one after the other in a single trajectory while also adequately identifying the correct bin associated with each object. Success is counted only when the robot can accurately sort *both* of the objects into their corresponding bins.

Offline pre-training dataset. The dataset provided for offline pre-training only consists of demonstrations that show how the robot should pick one of the two objects and place it into one of the two bins. Each episode in the pre-training dataset is about 30-40 timesteps long. A picture showing some trajectories from the pre-training dataset is shown in Figure 45. While the downstream task only requires solving this sorting task with two specific objects (shown in Figure 46), the pre-training data consists of 10 unique objects (some shown in Figure 45). The two target objects that appear together in the downstream target scene are never seen together in the pre-training data. Since the pre-training data only demonstrates how the robot must pick up one of the objects and place it in one of the two bins (not necessarily in the target bin that the target task requires), it neither consists of any behavior that places objects into bins sequentially nor does it consist of

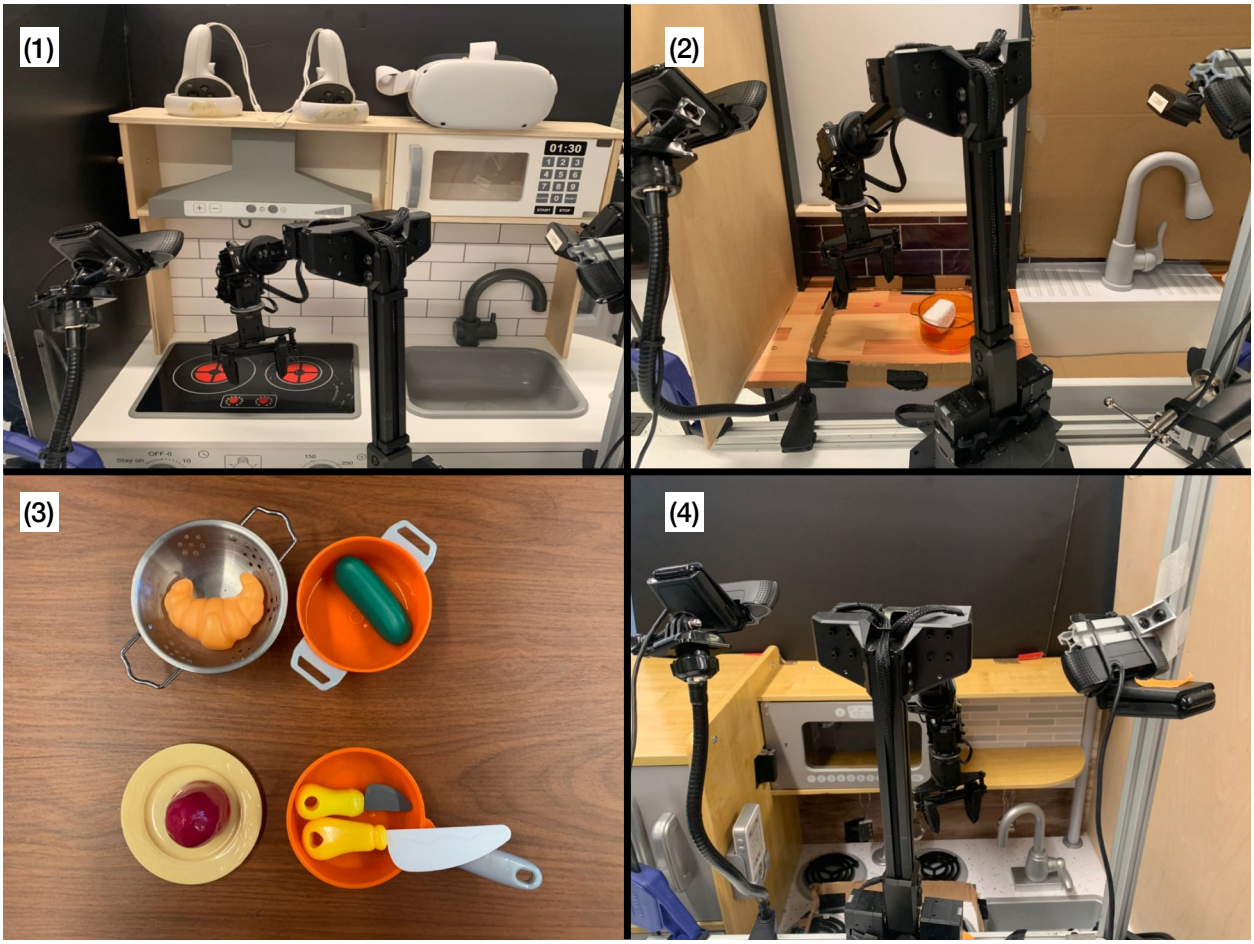


Figure 43: Setup Overview: Following Ebert et al. [70], we use a toykitchen setup described in that prior work for our experiments. This utilizes a 6-DoF WidowX 250 robot. **(1):** Held-out toykitchen used for experiments in Scenario 3 (denoted "toykitchen 6"), **(2):** Re-targeting toykitchen used for experiments in Scenario 2 (denoted "toykitchen 2"), **(3):** target objects used in the experiments of scenario 3., **(4):** the held-out kitchen setup used for door opening ("toykitchen 1").

any behavior where one of the objects is placed one of the bins while the other one is not. This is what makes this task particularly challenging.

Target demonstration data. The target task data provided to the algorithm consists of only ***five*** demonstrations that show how the robot must complete both the stages of placing both objects (see Figure 46). Each episode in the target demonstration data is 80 timesteps long, which is substantially longer than any trajectory in the pre-training data, though one would hope that good representations learned from the pick and place tasks are still useful for this target task. While all methods are able to generally solve the first segment of placing the first object into the correct bin, the primary challenge in this task is to effectively sort the second object, and we find that PTR attains a substantially better success rate than other baselines in this exact step.

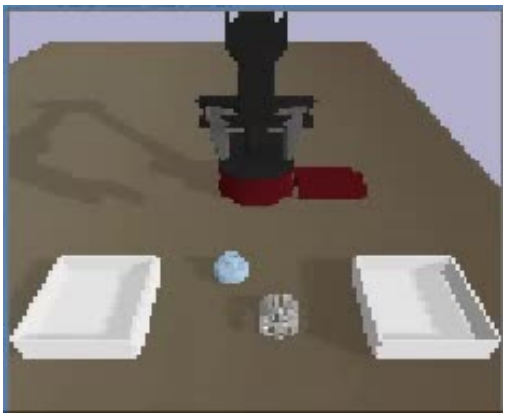


Figure 44: Bin-Sorting task used for our simulated evaluations. The task requires sorting the cylinder into the left bin and the teapot into the right bin.

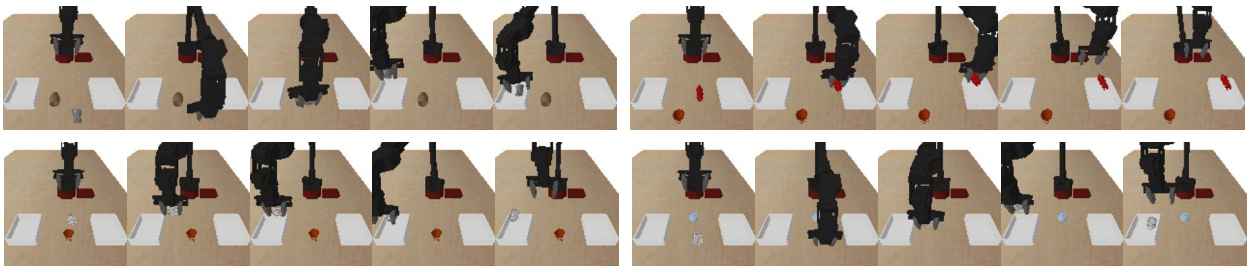


Figure 45: Some trajectories from the pre-training data used in the simulated bin-sort task.

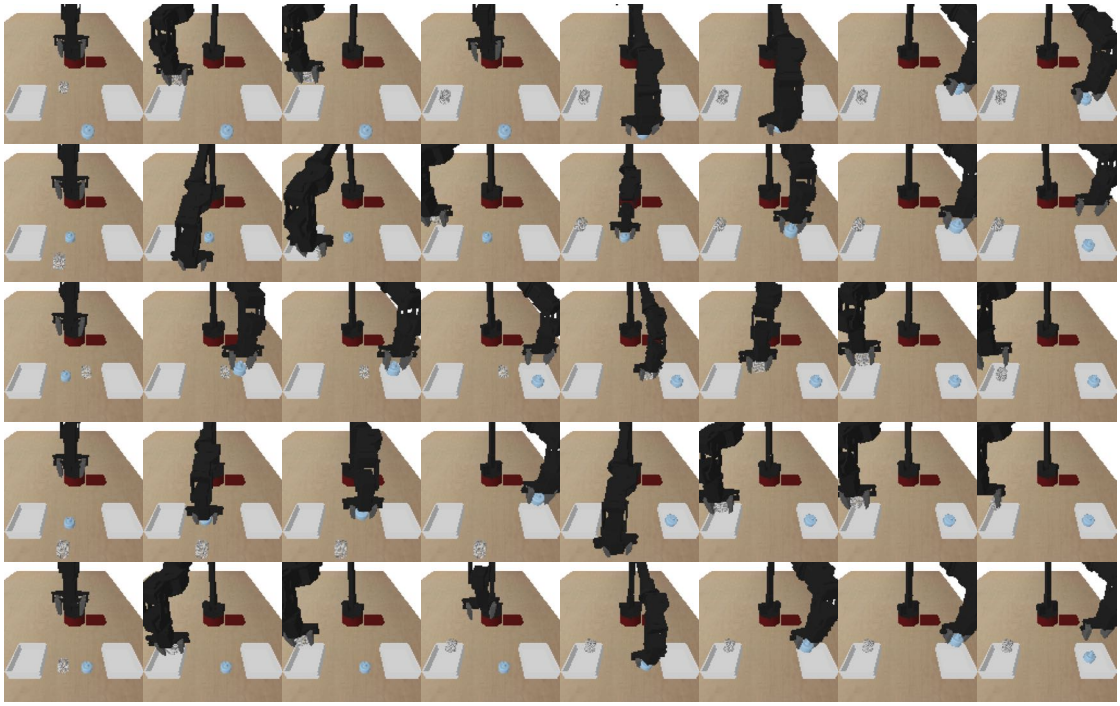


Figure 46: The five demonstration trajectories used for Phase 2 of PTR .

G.3 DESCRIPTION OF THE REAL-WORLD EVALUATION SCENARIOS

In this section, we describe the real-world evaluation scenarios considered in Section 9.4. We additionally include a much more challenging version of Scenario 3, for which we present results in Appendix G.4. These harder test cases evaluate the fine-tuning performance on four different tasks, starting from the same initialization trained on bridge data except the toykitchen 6 domain in which these four tasks were set up. In the following sections, the nomenclature for the toy kitchens is drawn from Ebert et al. [70] and as described in the caption of Figure 43.

G.3.1 *Scenario 1: Re-targeting skills for existing to solve new tasks*

Pre-training data. The pre-training data comprises all of the pick and place data from the bridge dataset [70] from toykitchen 2. This includes data corresponding to the task of putting the sushi in the transparent orange pot (Figure 47).

Target task and data. Since our goal in this scenario is to re-target the skill for putting the sushi in the transparent orange pot to the task of putting the sushi in the metallic pot, we utilize a dataset of 20 demonstrations that place the sushi in a metallic pot as our target task data that we fine-tune with (shown in Figure 47).

Figure 47: Illustration of pre-training data and finetuning data used for Scenario 1: re-targeting the put sushi in metal-pot behavior to put the object in the metal pot instead of the orange transparent pot.

Quantitative evaluation protocol. For our quantitative evaluations in Table 22, we run 10 controlled evaluation rollouts that place the sushi and the metallic pot in different locations of the workspace. In all runs, the arm starts at up to 10 cm distance above the target object. The initial object and arm poses and positions are matched as closely as possible for different methods.

G.3.2 *Scenario 2: Generalizing to Previously Unseen Domains*

Pre-training data. The pre-training data in Scenario 2 consists of 800 door-opening demonstrations on 12 different doors across 3 different toykitchen domains.

Target task and data. The target task requires opening the door of an unseen microwave in toykitchen 1 using a target dataset of only 15 demonstrations.

Quantitative evaluation protocol. We run 20 rollouts with each method, counting successes when the robot opened the door by at least 45 degrees. To perform this successfully, there is a degree of complexity as the robot has to initially open the door till it's open to about 30 degrees. Then due to physical constraints, the robot needs to wrap around the door and push it open from the inside. To begin an evaluation rollout, we reset the robot to randomly sampled poses obtained from held-out demonstrations on the target door. This is a compound task requiring the robot to first grab the door by the handle, next move around the door, and finally push the door open. As before, we match the initial pose of the robot as closely as possible for all the methods.

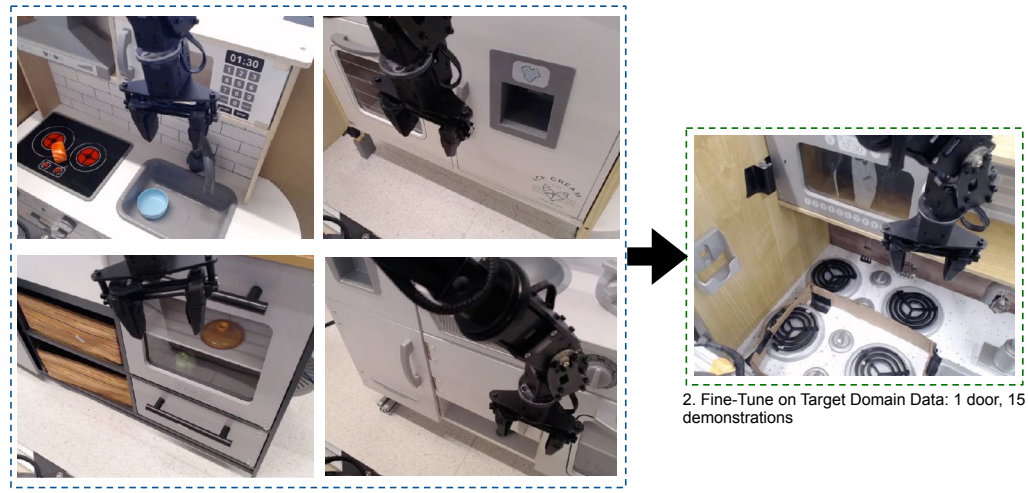


Figure 48: Illustration of pre-training data and fine-tuning data used for Scenario 2 (door opening): transferring a behavior to a held-out domain.

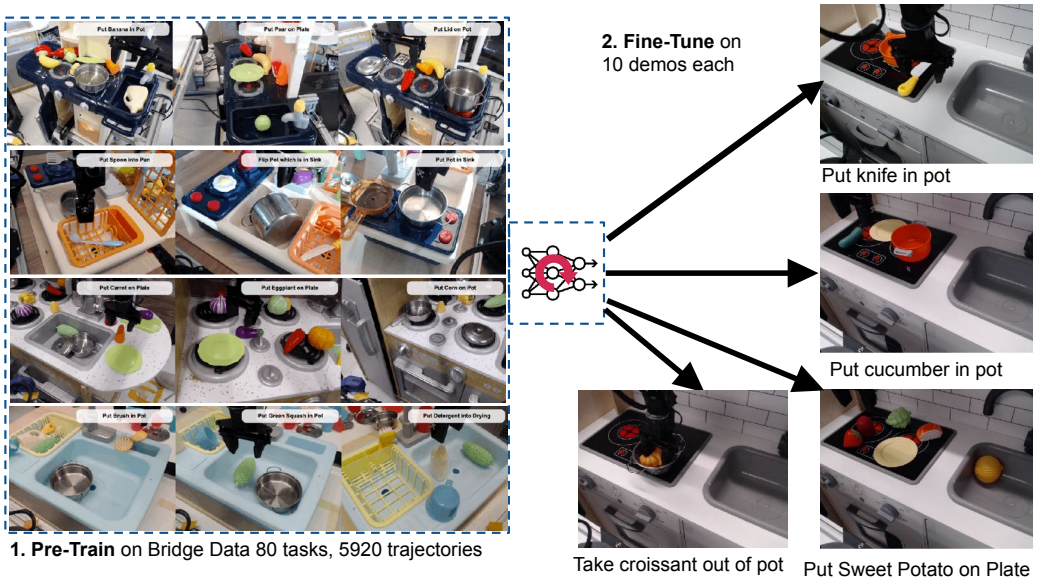


Figure 49: Illustration of pre-training data and fine-tuning data used for the new tasks we have added in Scenario 3. The goal is to learn to solve new tasks in new domains starting from the same pre-trained initialization and when fine-tuning is only performed using 10-20 demonstrations of the target task.

G.3.3 *Scenario 3: Learning to Solve New Tasks in New Domains*

Pre-training data. All pick-and-place data in the bridge dataset [70] except any demonstration data collected in toykitchen 6, where our evaluations are performed.

Target task and data. The target task requires placing corn in a pot in the sink in the new target domain and the target dataset provides 10 demonstrations for this task. These target demonstrations are sampled from the bridge dataset itself.

Quantitative evaluation protocol. During the evaluation we were unable to exactly match the camera orientation used to collect the target demonstration trajectories, and therefore ran evaluations with a slightly modified camera view. This presents an additional challenge for any method as it must now generalize to a modified camera view of the target toykitchen domain, without having ever observed this domain or this camera view during training. We sampled initial poses for our method by choosing transitions from a held-out dataset of demonstrations of the target task and resetting the robot to those initial poses for each method. We attempted to match the positions of objects across methods as closely as possible.

G.3.4 *More Tasks in Scenario 3: Learning to Solve Multiple New Tasks in New Domains From the Same Initialization*

In Appendix G.4, we have now added results for more tasks in Scenario 3. The details of these tasks are as follows:

Pre-training data. All pick-and-place data from bridge dataset [70] except data from toykitchen 6.

Target task and data. We consider four downstream tasks: take croissant from a metallic bowl, put sweet potato on a plate, place the knife in a pot, and put cucumber in a bowl. We collected 10 target demonstrations for the croissant, sweet potato, and put cucumber in bowl tasks, and 20 target demonstrations for the knife in pot task. A picture of these target tasks is shown in Figure 49.

Qualitative evaluation protocol. For our evaluations, we utilize either 10 or 20 evaluation rollouts. As with all of our other quantitative results, we evaluate all the baseline approaches and PTR starting from an identical set of initial poses for the robot. These initial poses are randomly sampled from the poses that appear in the first 10 timesteps of the held-out demonstration trajectories for this target task. For the configuration of objects, we test our policies in a variety of task-specific configurations that we discuss below:

- **Take croissant from metallic bowl:** For this task, we alternate between two kinds of positions for the metallic bowl. In the "easy" positions, the metallic bowl is placed

roughly vertically beneath the robot's initial starting pose, whereas in the "hard" positions, the robot must first move itself to the right location of the bowl and then execute the policy.

- **Put the cucumber in bowl:** We run 10 evaluation rollouts starting from 10 randomly sampled initial poses of the robot for our evaluations. Here we moved the bowl between the two stovetops in each trial.
- **Put sweet potato on plate:** For this task, we performed 20 evaluation rollouts. We only sampled 10 initial poses for the robot, but for each position, we evaluated every policy on two orientations of the sweet potato (i.e., the sweet potato is placed on the table on its flat face or on its curved face). Each of these orientations presents some unique challenges, and evaluating both of them allows us to gauge how robust the learned policy is to changes in orientation. The demonstration data had a variety of orientations for the sweet potato object that differed for each collected trajectory.
- **Place knife in pot:** We evaluate this task over 10 evaluation rollouts, where the first five rollouts use a smaller knife, while the other five rollouts use a larger knife (shown in Figure 43). Each knife was seen in the demonstration dataset with equal probability.

We will discuss the results obtained on these new tasks in Appendix G.4.

G.4 ADDITIONAL EXPERIMENTAL RESULTS

Figure 50: Sample observations from different camera viewpoints, only used during fine-tuning. Left: the original camera viewpoint found in Figure 49. **Middle:** an elevated camera viewpoint where the robot and camera have been raised 7 cm. **Right:** a rotated camera viewpoint where the kitchen has been slightly translated and rotated 15 degrees counterclockwise relative to the camera and robot.

Finetuning to novel camera viewpoints: Even though Scenario 3 already presents a novel toy-kitchen domain and previously unseen objects during finetuning, we also evaluate PTR on a more challenging scenario where we additionally alter the camera viewpoint during finetuning. We apply two kinds of alterations to the camera: **(a)** we elevate the mounting platform of the camera by 7 cm, which necessitates adapting the way the

physical coordinates of the robot end-effector are interpreted by the policy, and **(b)** we rotate the camera by about 15 degrees to induce a more oblique image observation than what was ever seen during pre-training. Note that in both of these scenarios, the robot has never encountered such camera viewpoints during pre-training, which makes this scenario even more challenging. The original dataset in [70] had the camera elevated to the same position for each domain and always ensured the kitchen was parallel to the camera platform, with translations being the primary changes in the scene for each domain. In Table 63, we present our results comparing PTR and BC (finetune). Observe that PTR still clearly outperforms BC (finetune), and attains performance close to that of PTR in Table 24, indicating that such shifts in the camera do not drastically hurt PTR .

Method	**Elevated Viewpoint**	**Rotated Viewpoint**
BC (finetune)	2/10	3/10
PTR (Ours)	**6/10**	**7/10**

Table 63: Comparison of PTR and BC (finetune), when evaluated on novel camera viewpoints with elevated and rotated cameras as shown in Figure 50 for the croissant task. Observe that PTR still outperforms BC (finetune) in this setting and attains more than 2x success rate of BC (finetune).

G.4.1 *Hyperparameters for PTR and Baseline Methods*

In this section, we will present the hyperparameters we use in our experiments and explain how we tune the other hyperparameters for both our method PTR and the baselines we consider.

PTR . Since PTR utilizes CQL as the base offline RL method, it trains two Q-functions and a separate policy, and maintains a delayed copy of the two Q-functions, commonly referred to as target Q-functions. We utilize completely independent networks to represent each of these five models (2 Q-functions, 2 target Q-functions, and the policy). We also do not share the convolutional encoders among them. As discussed in the main text, we rescaled the action space to $[-1, 1]^{|\mathcal{A}|}$ to match the one used by actor-critic algorithms, and utilized a Tanh squashing function at the end of the policy. We used a CQL α value of 10.0 for our pick-and-place experiments. The rest of the hyperparameters for training the Q-function, the target network updates, and the policy are taken from the standard training for image-based CQL from Singh et al. [303] and are presented in Table 65 below for completeness. The hyperparameters we choose are essentially the network design decisions of **(1)** utilizing group normalization instead of batch normalization, **(2)** utilizing learned spatial embeddings instead of standard mean pooling, **(3)** passing in actions at each of the fully connected layers of the Q-network and the hyperparameter α in CQL that must be adjusted since our data consists of demonstrations. We will ablate the new design decisions explicitly in Appendix G.5.

Hyperparameter	Value
Q-function learning rate	3e-4
Policy learning rate	1e-4
Target update rate	0.005 (soft update with Polyak averaging)
Optimizer type	Adam
Discount factor γ	0.96 (since trajectories have a length of only about 30-40)
Use terminals	True
Reward shift and scale	shift = -1, scale = 10.0
CQL α	10.0
Use Color Jitter	True
Use Random Cropping	True

Table 64: Main hyperparameters for CQL training in our real-world experiments. In the simulation, we utilize a smaller α for CQL, $\alpha = 1.0$, and a larger discount $\gamma = 0.98$ since trajectories in the simulation are about 60-70 timesteps in length.

The only other hyperparameter used by PTR is the mixing ratio τ that determines the proportion of samples drawn from the pre-training dataset and the target dataset during the offline finetuning phase in PTR . We utilize $\tau = 0.7$ for our experiments with PTR in the main paper, and use $\tau = 0.9$ for the additional experiments we added in the Appendix. This is because $\tau = 0.9$ (more bridge data, and a smaller amount of target data) was helpful in scenarios with very limited target data.

In order to perform checkpoint selection for PTR , we utilized the trends in the learned Q-values over a set of held-out trajectories on the target data as discussed in Section 9.3.2. We did not tune any other algorithmic hyperparameters for CQL, as these were taken directly from [303].

BC (finetune). We trained BC in a similar manner as Ebert et al. [70], utilizing the design decisions that this prior work found optimal for their experiments. The policy for BC utilizes the very same ResNet 34 backbone as our RL policy since a backbone based on ResNet 34 was found to be quite effective in Ebert et al. [70]. Following the recommendations of Ebert et al. [70] and based on result trends from our own preliminary experiments, we chose to not utilize the tanh squashing function at the end of the policy for any BC-based method, but trained a deterministic BC policy that was trained to regress to the action in the demonstration with a mean-squared error (MSE) objective.

In order to perform cross-validation, checkpoint, and model selection for our BC policies, we follow guidelines from prior work [70, 71] and track the MSE on a held-out validation dataset similar to standard supervised learning. We found that a ResNet 34 BC policy attained the smallest validation MSEs in general, and for our evaluations, we utilized a checkpoint of a ResNet 34 BC policy that attained the smallest MSE.

Hyperparameter	Value
Policy learning rate	1e-4
Optimizer type	Adam
Use Color Jitter	True
Use Random Cropping	True
Dropout	0.4

Table 65: Main hyperparameters for Behavior Cloning Baseline Training in our real-world and simulation experiments. Note: architecture design choices follow closely to PTR design choices.

Analogous to the case of PTR discussed above, we also ablated the performance of BC for a set of varying values of the mixing ratio τ, but found that a large value of $\tau = 0.9$ was the most effective for BC, and hence utilized $\tau = 0.9$ for BC (finetune) and BC (joint).

BC (joint) and CQL (joint). The primary distinction between training **BC (joint)** and **BC (finetune)** and correspondingly, **CQL (joint)** and PTR was that in the case of joint training, the target dataset was introduced right at the beginning of Phase 1 (pre-training phase), and we mixed the target data with the pre-training data using the same value of the mixing ratio τ used in for our fine-tuning experiments to ensure a fair comparison.

Few-shot offline meta-RL (MACAW) [230]: We compare to two variants of this algorithm and perform an **extensive** sweep over several hyperparameters, shown in Table 66. We trained two different variants of MACAW in our evaluation: **(1)** Pre-training on the bridge data in Scenario 3 and then fine-tuning on target data of interest, and **(2)** adapting a set of existing task identifiers to the target task of interest utilizing the same pre-training and fine-tuning domains. We performed early stopping on the meta-training based on validation losses. From there, we started the meta-testing phase, adapting to the target domain of interest. Following Mitchell et al. [230], we use a task mini-batch of 8 tasks at each step of optimization rather than using all of the training tasks. We clipped the advantage weight logits to the scale of 20 and attempted to utilize a policy network with a fixed and learned standard deviation. Additionally, we varied the number of Adaptation steps following prior work. Our evaluation protocol for MACAW entails utilizing the validation losses to choose an initial checkpoint for evaluation. Then, we consider checkpoints in the neighborhood ($\pm$ 50K gradient steps) to for evaluations as well and chose the max over all of these checkpoints as the final evaluation success rate.

Quantitatively, as seen in Table 24, MACAW was unable to get non-zero success rates on any of the tasks we study. However, we did qualitatively observe nontrivial behavior seen in our evaluation rollouts. For instance, we found that the policies trained via MACAW could consistently grasp the object of interest but were unable to localize where to place the object correctly. Several trials involved hovering around with the object of interest

and not placing the object in the container. Other trials involved the agent failing to grasp the object.

Hyperparameter	Value
Optimizer	Adam
Outer Policy learning rate	1e-4
Outer Value learning rate	1e-5, 1e-6
Inner Policy learning rate	1e-2, 1e-3
Inner Value learning rate	1e-3, 1e-4
Auxilary Advantage Coefficient	1e-2, 1e-3, 1e-4
Policy Parameterization	Fixed std, Learned std
AWR Policy Temperature	1, 10, 20
Number of Adaptation Steps	1, 2, 3
Task Batch Size	8
Train Adaptation Batch Size	64
Eval Adaptation Batch Size	64
Max Advantage Clip	20
Use Color Jitter	True
Use Random Cropping	True

Table 66: Main hyperparameters for Training MACAW [230] in our real-world experiments. Note: architecture design choices follow closely to PTR design choices but hyperparameter design choices follow closely the suggestions in Mitchell et al. [230].

Pre-trained R3M initialization [243]: Next we compare PTR to utilizing an off-the-shelf pre-trained representation given by R3M [243]. We compare two baselines that attempt to train an MLP policy on top of the R3M state representation by using BC (finetuning) and CQL (finetuning) respectively. To ensure that this baseline is well-tuned, we tried a variety of network sizes with 2, 3 or 4 MLP layers and also tuned the hidden dimension sizes in [256, 512, 1024]. We also utilized dropout as regularization to prevent overfitting and tuned a variety of values of dropout probability in [0, 0.1, 0.2, 0.4, 0.6, 0.8]. We observe in Table 24, that on the four tasks we evaluate on, PTR outperforms R3M, which indicates that training on the bridge dataset can indeed give rise to effective visual representations that are more suited to finetuning in our setting. The numbers we report in the table are the best over each parametric policy corresponding to each hyperparameter in our ablation. Checkpoint selection was done utilizing early stopping which is the last iteration where the validation error stops decreasing.

Pre-trained MAE initialization [130]: We took a similar training procedure to R3M for our MAE representation. We used an MAE trained on every image from the bridge dataset Ebert et al. [70]. We then fine-tuned a specific target task with a similar ablation on network size, hidden dimension size, and regularization techniques such as dropout. We observe in Table 25, that on the four tasks we evaluate on, PTR outperforms R3M,

which indicates that training on the bridge dataset can indeed give rise to effective visual representations that are more suited to finetuning in our setting. The numbers we report in the table are the best over each parametric policy corresponding to each hyperparameter in our ablation. Checkpoint selection was done utilizing early stopping which is the last iteration where the validation error stops decreasing.

Policy expressiveness study. We considered two policy expressiveness choices for BC to compare with our reference BC implementation that is implemented with a set of MLP layers. The first of the two choices was an **autoregressive policy** where the 7-dimensional action space was discretized into 100 bins. Each action was then predicted autoregressively conditioned on the observation, task id, and the action component from the previous dimension(s). The second approach was with the BeT Architecture from Shafiullah et al. [291]. We utilized the reference implementation from the paper with the default suggested hyperparameters for this set of ablations. The window size for the MinGPT transformer was ablated over between 1, 2, and 10.

G.5 VALIDATING THE DESIGN CHOICES FROM SECTION 9.3.2

In this section, we will present ablation studies aimed to validate the design choices utilized by PTR . We found these design choices quite crucial for attaining good performance. The concrete research questions we wish to answer are: **(1)** How effective is a learned spatial embedding compared to other approaches for aggregating spatial information? **(2)** Is concatenating actions at each fully-connected layer of the Q-function crucial for good performance?, **(3)** Is group normalization a good alternative to batch normalization? and **(4)** How does our choice of creating binary rewards for training affect the performance of PTR ?. We will answer these questions next.

Learned spatial embeddings are crucial for performance. Next we study the impact of utilizing the learned spatial embeddings for encoding spatial information when converting the feature maps from the convolutional stack into a vector that is fed into the fully-connected part of the Q-function. We compare our choice to utilizing a spatial softmax as in Ebert et al. [70], and also global average pooling (GAP) that simply averages over the spatial information, typically utilized in supervised learning with ResNets.

Method	**Success rate**
PTR with spatial softmax	4/10
PTR with global average pooling	4/10
PTR with learned spatial embeddings **(Ours)**	**7/10**

Table 67: Ablation of PTR with spatial softmax and GAP on the croissant task. Note that PTR with learned spatial embeddings performs significantly better than using a spatial softmax or average pooling.

As shown in Table 67 learned spatial embeddings outperform both of these prior approaches on the put croissant in pot task. We suspect that spatial softmax does not perform much better than the GAP approach since the softmax operation can easily get saturated when running gradient descent to fit value targets that are not centered in some range, which would effectively hinder its expressivity. This indicates that the approach of retaining spatial information like in PTR is required for attaining good performance.

Concatenating actions at each layer is crucial for performance. Next, we run PTR without passing in actions at each fully connected layer of the Q-function on the take croissant out of metallic bowl task and only directly concatenate the actions with the output of the convolutional layers before passing it into the fully-connected component of the network. On the croissant task, we find that not passing in actions at each layer only succeeds in **2/10** evaluation rollouts, which is significantly worse than the default PTR which passes in actions at each layer and succeeds in **7/10** evaluation rollouts (Table 68).

Method	Success rate
PTR without actions passed in at each FC layer	2/10
PTR with actions passed in at each FC layer (Ours)	**7/10**

Table 68: Ablation of PTR with actions passed in at each layer. Observe that passing in actions at each fully-connected layer does lead to quite good performance.

Group normalization is more consistent than batch normalization. Next, we ablate the usage of group normalization over batch normalization in the ResNet 34 Q-functions that PTR uses. We found that batch normalization was generally harder to train to attain Q-function plots that exhibit a roughly increasing trend over the course of a trajectory. That said, on some tasks such as the croissant in pot task, we did get a reasonable Q-function, and found that batch normalization can perform well. On the other hand, on the put cucumber in pot task, we found that batch normalization was really ineffective. These results are shown in Table 69, and they demonstrate that batch normalization may not be as consistent and reliable with PTR as group normalization.

Method	Croissant out of metallic bowl	Cucumber in pot
PTR with batch norm. (relative)	+ 28.0% (7/10 $\rightarrow$ 9/10)	- 60.0% (5/10 $\rightarrow$ 2/10)

Table 69: Relative performance of PTR with batch normalization with respect to PTR with group normalization. Observe that while utilizing batch normalization in PTR can be sometimes more effective than using group normalization (e.g., take croissant out of metallic bowl task), it may also be highly ineffective and can reduce success rates significantly in other tasks. The performance numbers to the left of the $\rightarrow$ correspond to the performance of PTR with group normalization and the performance to the right of $\rightarrow$ is the performance with batch normalization.

Choice of the reward function. Finally, we present some results that ablate the choice of the reward function utilized for training PTR from data that entirely consists of demonstrations. In our main set of experiments, we labeled the last three timesteps of every trajectory with a reward of +1 and annotated all other timesteps with a 0 reward. We tried perhaps the most natural choice of labeling only the last timestep with a 0 reward on the croissant task and found that this choice succeeds **0/10** times, compared to annotating the last three timesteps with a +1 reward which succeeds **7/10** times. We suspect that this is because only annotating the last timestep with a +1 reward is not ideal for two reasons: first, the task is often completed in the dataset much earlier than the observation shows the task complete, and hence the last-step annotation procedure induces a non-Markovian reward function, and second, only labeling the last step with a +1 leads to overly conservative Q-functions when used with PTR , which may not lead to good policies.

G.6 MORE DETAILS FOR ONLINE FINE-TUNING

Offline pre-training. For both PTR and BC baseline, we used 40 open-door demonstrations as target task data and combined them with the Bridge Dataset to pre-train the policy. To reduce the training time in the real system, we used ResNet 18 backbones.

Reset policy. For the reset policy, we additionally collected 22 close-door demonstrations as the target task data and pre-trained the policy with PTR. Similar to the open-door policy, we used ResNet 18 backbones to save training time.

Reward classifier. We used a ResNet 34 classification model and trained it to detect whether the door is open or closed from visual inputs. For the training data, we manipulated the robot to collect around 20 positive and negative trajectories for both open and closed doors.

Method. As discussed in Chapter 8, offline value function initializations from CQL may not be effective at fine-tuning if the learned Q-values are not at the same scale as the ground-truth return of the behavior policy. While this property does not affect offline performance, it is crucial to enforce this property during fine-tuning. That said, this property can be "baked in" by simply preventing the CQL regularizer from minimizing the learned Q-function if its values fall below the Monte-Carlo return of the trajectories in the dataset. Therefore, for the online fine-tuning experiment, we utilize Cal-QL incorporated into the offline CQL approach.

Hyperparameters. For both online fine-tuning with PTR and SACfD, we performed the experiment by mixing the Bridge Dataset, offline target data, and the online data in a ratio (β) of 0.35, 0.15, and 0.5. For PTR , we used the CQL alpha value of 5 for the offline phase and 0.5 for the online phase.

Figure 51: Example of unsafe behaviors when running SACfD. The robot collides with the camera during online exploration, resulting in a system crash.

Evaluation. The results shown in Figure 8 were evaluated autonomously every 5K environment step during the online fine-tuning. Each evaluation was assessed with 10 trials, one from each initial position. The results shown in Figure 27 were additionally evaluated over 3 trials from each initial position, using the offline initialization and the final checkpoint obtained after 20K environment steps of online fine-tuning.

H

APPENDIX: HARDWARE ACCELERATOR DESIGN

H.1 ADDITIONAL EXPERIMENTS

In this section, we present some additional results obtained by jointly optimizing multiple applications (Appendix H.1.2), provide an analysis of the designed accelerators (Appendix H.1.4) and finally, discuss how our trained conservative surrogate can be used with a different evaluation time constraint (Appendix H.1.3).

H.1.1 *Comparison to Other Baseline Methods*

Comparison to P3BO. We perform a comparison against P3BO, a state-of-the-art online method in biological sequence design [12]. On average, PRIME outperforms the P3BO method by 2.5× (up to 8.7× in U-Net). In addition, we present the comparison between the total simulation runtime of the P3BO and Evolutionary methods in Figure 52. Note that, not only the total simulation time of P3BO is around 3.1× higher than the Evolutionary method, but also the latency of final optimized accelerator is around 18% for MobileNetEdgeTPU. On the other hand, the total simulation time of PRIME for the task of accelerator design for MobileNetEdgeTPU is lower than both methods (only 7% of the Evolutionary method as shown in Figure 14).

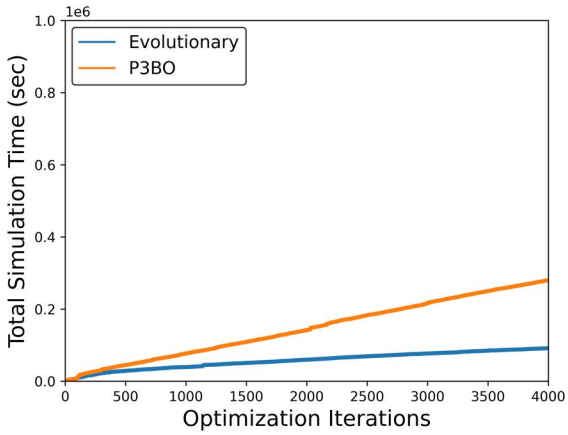


Figure 52: Comparing the total simulation time needed by the P3BO and Evolutionary method on MobileNetEdgeTPU. Note that, not only the total simulation time of P3BO is around 3.1× higher than the Evolutionary method, but also the latency of final optimized accelerator is around 18% for MobileNetEdgeTPU. The total simulation time of our method is around 7% of the Evolutionary method (See Figure 14).

Table 70: Optimized objective values (i.e., latency in milliseconds) obtained by PRIME and P3BO [12] when optimizing over single applications (MobileNetEdgeTPU, M4, t-RNN Dec, t-RNN Enc, and U-Net). On average, PRIME outperforms P3BO by 2.5×.

Application	**PRIME (Ours)**	**P3BO**
MobileNetEdgeTPU	**298.50**	376.02
M4	**370.45**	483.39
U-Net	**740.27**	771.70
t-RNN Dec	**132.88**	865.12
t-RNN Enc	**130.67**	1139.48
Geomean of PRIME 's Improvement	1.0×	2.5×

H.1.2 *Learned Surrogate Model Reuse for Accelerator Design*

Extending our results in Table 31, we present another variant of optimizing accelerators jointly for multiple applications. In that scenario, the learned conservative model is reused to architect an accelerator for a subset of applications used for training. We train a contextual conservative model on the variants of MobileNet (Table 29) as discussed in Section 10.4, but generated optimized designs by only optimizing the average surrogate on only two variants of MobileNet (MobileNetEdgeTPU and MobileNetV2). This tests the ability of our approach PRIME to provide a general contextual conservative surrogates, that can be trained *only once* and optimized multiple times with respect to different subsets of applications. Observe in Table 71, PRIME architects high-performing accelerator

Table 71: Optimized objective values (i.e., latency in milliseconds) obtained by our PRIME when using the jointly optimized model on three variants of MobileNets and use for MobileNetEdgeTPU and MobileNetV2 for different dataset configurations. PRIME outperforms the best online method by **7%** and finds an accelerator that is **3.29**× better than the best accelerator in the training dataset (last row). The best accelerator configuration is highlighted in bold.

	PRIME				Online Optimization		
Applications	**All**	**-Opt**	**-Infeasible**	**Standard**	**Bayes Opt**	**Evolutionary**	$\mathcal{D}$ **(Best in Training)**
(MobileNetEdgeTPU, MobileNetV2)	**253.85**	297.36	264.85	341.12	275.21	271.71	834.68

Table 72: Optimized objective values (i.e., latency in milliseconds) obtained by various methods for the task of learning accelerators specialized to MobileNetEdgeTPU under chip area budget constraint 18 mm^2 reusing the already learned model by our method for MobileNetEdgeTPU (shown in Table 30). Lower latency/runtime is better. From left to right: our method, our method without negative sampling ("PRIME -Opt") and without utilizing infeasible points ("PRIME -Infeasible"), standard surrogate ("Standard"), online Bayesian optimization ("Bayes Opt"), online evolutionary algorithm ("Evolutionary") and the best design in the training dataset. Note that PRIME improves over the best in the dataset by 12%, outperforms the best online optimization method by 4.4%. The best accelerator configuration is highlighted in bold.

	PRIME				Online Optimization		
Applications	**All**	**-Opt**	**-Infeasible**	**Standard**	**Bayes Opt**	**Evolutionary**	$\mathcal{D}$ **(Best in Training)**
MobileNetEdgeTPU, Area $\leq$ 18 mm^2	**315.15**	433.81	351.22	470.09	331.05	329.13	354.13

configurations (better than the best point in the dataset by 3.29× – last column) while outperforming the online optimization methods by 7%.

H.1.3 *Learned Surrogate Model Reuse under Different Design Constraint*

We also test the robustness of our approach in handling variable constraints at test-time such as different chip area budget. We evaluate the learned conservative surrogate trained via PRIME under a reduced value of the area threshold, α, in Equation 10.3.1. To do so, we utilize a variant of rejection sampling – we take the learned model trained for a default area constraint $\alpha = 29$ mm^2 and then reject all optimized accelerator configurations which do not satisfy a reduces area constraint: $\text{Area}(\boldsymbol{x}) \leq \alpha_0 = 18$ mm^2. Table 72 summarizes the results for this scenario for the MobileNetEdgeTPU [121] application under the new area constraint ($\alpha = 18$ mm^2). A method that produces diverse designs which are both high-performing and are spread across diverse values of the area constraint are expected to perform better. As shown in Table 72, PRIME provides better accelerator than the best online optimization from scratch with the new constraint value by 4.4%, even when PRIME does not train its conservative surrogate with this unseen test-time design constraint. Note that, when the design constraint changes, online methods generally need to restart the optimization process from scratch and undergo costly queries to the simulator. This would impose additional overhead in terms of total simulation time (§ Figure 14 and Figure 15). However, the results in Table 72 shows that our learned

Table 73: Per application latency for the best accelerator design suggested by PRIME and the Evolutionary method according to Table 31 for multi-task accelerator design (nine applications and area constraint 100 mm^2). PRIME outperforms the Evolutionary method by 1.35×.

Applications	Latency (ms)		Improvement of PRIME over Evolutionary
	PRIME	Evolutionary (Online)	
MobileNetEdgeTPU	**288.38**	319.98	1.10×
MobileNetV2	**216.27**	255.95	1.18×
MobileNetV3	**487.46**	573.57	1.17×
M4	**400.88**	406.28	1.01×
M5	248.18	**239.18**	0.96×
M6	164.98	**148.83**	0.90×
U-Net	1268.73	**908.86**	0.71×
t-RNN Dec	**191.83**	862.14	5.13×
t-RNN Enc	**185.41**	952.44	4.49×
Average (Latency in ms)	383.57	518.58	1.35×

Table 74: Optimized accelerator configurations (See Table 28) found by PRIME and the Evolutionary method for multi-task accelerator design (nine applications and area constraint 100 mm^2). Last row shows the accelerator area in mm^2. PRIME reduces the overall chip area usage by 1.97×. The difference in the accelerator configurations are shaded in gray.

Accelerator Parameter	Parameter Value	
	PRIME	Evolutionary (Online)
# of PEs-X	4	4
# of PEs-Y	6	8
# of Cores	64	128
# of Compute Lanes	4	6
PE Memory	2,097,152	1,048,576
Core Memory	131,072	131,072
Instruction Memory	32,768	8,192
Parameter Memory	4,096	4,096
Activation Memory	512	2,048
DRAM Bandwidth (Gbps)	30	30
Chip Area (mm^2)	46.78	92.05

surrogate model can be reused under different test-time design constraint eliminating additional queries to the simulator.

H.1.4 *Analysis of Designed Accelerators*

In this section, we overview the best accelerator configurations that PRIME and the Evolutionary method identified for multi-task accelerator design (See Table 31), when

the number of target applications are nine and the area constraint is set to 100 mm^2. The average latencies of the best accelerators found by PRIME and the Evolutionary method across nine target applications are **383.57 ms** and **518.58 ms**, respectively. In this setting, our method outperforms the best online method by **1.35**×. Table 73 shows per application latencies for the accelerator suggested by our method and the Evolutionary method. The last column shows the latency improvement of PRIME over the Evolutionary method. Interestingly, while the latency of the accelerator found by our method for MobileNetEdgeTPU, MobileNetV2, MobileNetV3, M4, t-RNN Dec, and t-RNN Enc are better, the accelerator identified by the online method yields lower latency in M5, M6, and U-Net.

To better understand the trade-off in design of each accelerator designed by our method and the Evolutionary method, we present all the accelerator parameters (See Table 28) in Table 74. The accelerator parameters that are different between each of the designed accelerator are shaded in gray (e.g. # of PEs-Y, # of Cores, # of Compute Lanes, PE Memory, Instruction Memory, and Activation Memory). Last row of Table 74 depicts the overall chip area usage in mm^2. PRIME not only outperforms the Evolutionary algorithm in reducing the average latency across the set of target applications, but also reduces the overall chip area usage by **1.97**×. Studying the identified accelerator configuration, we observe that PRIME trade-offs compute (**64 cores vs. 128 cores**) for larger PE memory size (**2,097,152 vs. 1,048,576**). These results show that PRIME favors PE memory size to accommodate for the larger memory requirements in t-RNN Dec and t-RNN Enc (See Table 29 Model Parameters) where large gains lie. Favoring larger on-chip memory comes at the expense of lower compute power in the accelerator. This reduction in the accelerator's compute power leads to higher latency for the models with large number of compute operations, namely M5, M6, and U-Net (See last row in Table 29). M4 is an interesting case where both compute power and on-chip memory is favored by the model (6.23 MB model parameters and 3,471,920,128 number of compute operations). This is the reason that the latency of this model on both accelerators, designed by our method and the Evolutionary method, are comparable (400.88 ms in PRIME vs. 406.28 ms in the online method).

H.1.5 *Comparison with Online Methods in Zero-Shot Setting*

We evaluated the Evolutionary (online) method under two protocols for the last two rows of Table 32: first, we picked the best designs (top-performing 256 designs similar to the PRIME setting in Section 10.4) found by the evolutionary algorithm on the training set of applications and evaluated them on the target applications and second, we let the evolutionary algorithm continue simulator-driven optimization on the target applications. The latter is unfair, in that the online approach is allowed access to querying more designs in the simulator. Nevertheless, we found that in either configuration, the evolutionary approach performed worse than PRIME which does not access training data from the

Table 75: Optimized objective values (i.e., latency in milliseconds) obtained by various methods for the task of learning accelerators specialized to a given application. Lower latency/runtime is better. From left to right: our method, our method without negative sampling ("PRIME -Opt") and without utilizing infeasible points ("PRIME -Infeasible"), standard surrogate ("Standard"), online Bayesian optimization ("Bayes Opt"), online evolutionary algorithm ("Evolutionary") and the best design in the training dataset. Note that, in all the applications PRIME improves over the best in the dataset, outperforms online optimization methods in 7/9 applications and the complete version of PRIME generally performs best. The best accelerator designs are in bold.

Application	PRIME			Standard	Online Optimization		$\mathcal{D}$ (Best in Training)
	All	-Opt	-Infeasible		Bayes Opt	Evolutionary	
MobileNetEdge	**298.50**	435.40	322.20	411.12	319.00	320.28	354.13
MobileNetV2	**207.43**	281.01	214.71	374.52	240.56	238.58	410.83
MobileNetV3	**454.30**	489.45	483.96	575.75	534.15	501.27	938.41
M4	**370.45**	478.32	432.78	1139.76	396.36	383.58	779.98
M5	208.21	319.61	246.80	307.57	201.59	**198.86**	449.38
M6	131.46	197.70	162.12	180.24	121.83	**120.49**	369.85
U-Net	**740.27**	740.27	765.59	763.10	872.23	791.64	1333.18
t-RNN Dec	**132.88**	172.06	135.47	136.20	771.11	770.93	890.22
t-RNN Enc	**130.67**	134.84	137.28	150.21	865.07	865.07	584.70

target application domain. For the area constraint 29 mm^2 and 100 mm^2, the Evolutionary algorithm reduces the latency from 1127.64 $\rightarrow$ 820.11 and 861.69 $\rightarrow$ 552.64, respectively, although still worse than PRIME . In the second experiment in which we *unfairly* allow the evolutionary algorithm to continue optimizing on the target application, the Evolutionary algorithm suggests worse designs than Table 32 (e.g. 29 mm^2: 1127.64 $\rightarrow$ 1181.66 and 100 mm^2: 861.69 $\rightarrow$ 861.66).

H.1.6 *PRIME Ablation Study*

Here we ablate over variants of our method: (1) Opt was not used for negative sampling ("PRIME -Opt" in Table 75) (2) infeasible points were not used ("PRIME -Infeasible" in Table 75). As shown in Table 75, the variants of our method generally performs worse compared to the case when both negative sampling and infeasible data points are utilized in training the surrogate model.

H.1.7 *Comparison with Human-Engineered Accelerators*

In this section, we compare the optimized accelerator design found by PRIME that is targeted towards single applications to the manually optimized EdgeTPU design [361, 121]. EdgeTPU accelerators are primarily optimized towards running applications in image classification, particularly, MobileNetV2, MobileNetV3 and MobileNetEdgeTPU. The goal of this comparison is to present the potential benefit of PRIME for a dedicated application when compared to human designs. For this comparison, we utilize an area

Table 76: The comparison between the accelerator designs suggested by PRIME and EdgeTPU [361, 121] for single model specialization. On average (last row), with single-model specialization our method reduces the latency by 2.69× while minimizes the chip area usage by 1.50×.

Application	Latency (milliseconds)			Chip Area (mm^2)		
	PRIME	EdgeTPU	Improvement	PRIME	EdgeTPU	Improvement
MobileNetEdgeTPU	294.34	523.48	1.78×	18.03	27	1.50×
MobileNetV2	208.72	408.24	1.96×	17.11	27	1.58×
MobileNetV3	459.59	831.80	1.81×	11.86	27	2.28×
M4	370.45	675.53	1.82×	19.12	27	1.41×
M5	208.42	377.32	1.81×	22.84	27	1.18×
M6	132.98	234.88	1.77×	16.93	27	1.59×
U-Net	1465.70	2409.73	1.64×	25.27	27	1.07×
t-RNN Dec	132.43	1384.44	10.45×	14.82	27	1.82×
t-RNN Enc	130.45	1545.07	11.84×	19.87	27	1.36×
Average Improvement	—	—	**2.69×**	—	—	**1.50×**

constraint of 27 mm^2 and a DRAM bandwidth of 25 Gbps, to match the specifications of the EdgeTPU accelerator.

Table 76 shows the summary of results in two sections, namely "Latency" and "Chip Area". The first and second under each section show the results for PRIME and EdgeTPU, respectively. The final column for each section shows the improvement of the design suggested by PRIME over EdgeTPU. On average (as shown in the last row), PRIME finds accelerator designs that are 2.69× (up to 11.84× in t-RNN Enc) better than EdgeTPU in terms of latency. Our method achieves this improvement while, on average, reducing the chip area usage by 1.50× (up to 2.28× in MobileNetV3). Even on the MobileNet image-classification domains, we attain an average improvement of 1.85×.

H.1.8 *Zero-Shot Results on All Applications*

In this section, we present the results of zero-shot optimization from Table 32 on all the nine applications we study in the paper (i.e., test applications = all nine models: MobileNet (EdgeTPU, V2, V3), M6, M5, M4, t-RNN (Enc and Dec), and U-Net). We investigate this for two sets of training applications and two different area budgets. As shown in Table 77, we find that PRIME does perform well compared to the online evolutionary method.

Table 77: Optimized objective values (i.e., latency in milliseconds) under zero-shot setting when the test applications include all the nine evaluated models (e.g. MobileNet (EdgeTPU, V2, V3), M4, M5, M6, t-RNN Dec, t-RNN Enc, U-Net). Lower latency is better. From **left** to **right**: the applications used to train the surrogate model in PRIME the target applications for which the accelerator is optimized for, the area constraint of the accelerator, PRIME 's (best, median) latency, and best online method's (best, median) latency. The best accelerator configurations identified is highlighted in bold.

Train Applications	Area	PRIME	Evolutionary (Online)
MobileNet (EdgeTPU,V2,V3), M4, M5, M6, t-RNN Enc	29 mm^2	(**426.65**, **427.94**)	(586.55, 586.55)
MobileNet (EdgeTPU,V2,V3), M4, M5, M6, t-RNN Enc	100 mm^2	(**365.95**, **366.64**)	(518.58, 519.37)
Geomean of PRIME 's Improvement	—	(1.0×, 1.0×)	(1.40×, 1.39×)

H.1.9 *Different Train and Validation Splits*

In the main paper, we used the worst 80% of the feasible points in the training dataset for training and used the remaining 20% of the points for cross-validation using our strategy based on Kendall's rank correlation. In this section, we explore some alternative training-validation split strategies to see how they impact the results. To do so, we consider two alternative strategies: **(1)** training on 95% of the worst designs, validation on top 5% of the designs, and **(2)** training on the top 80% of the designs and validation on the worst 20% of the designs. We apply these strategies to MobileNetEdgeTPU, M6 and t-RNN Enc models from Table 30, and present a comparative evaluation in Table 78 below.

Results. As shown in Table 78, we find that cross-validating using the best 5% of the points in the dataset led to a reduced latency (298.50 → 273.30) on MobileNetEdgeTPU, and retained the same performance on M6. However, it increased the latency on t-RNN Enc (130.67 → 137.45). This indicates at the possibility that while top 5% of the datapoints can provide a better signal for cross-validation in some cases, this might also hurt performance if the size of the 5% dataset becomes extremely small (as in the case of t-RNN Enc, the total dataset size is much smaller than either MobileNetEdgeTPU or M6).

The strategy of cross-validating using the worst 20% of the points hurt performance on M6 and t-RNN Enc, which is perhaps as expected, since the worst 20% of the points may not be indicative of the best points found during optimization. However, while it improves performance on the MobileNetEdgeTPU application compared to the split used in the main paper but it is still worse than using the top 5% of the points for validation.

Table 78: Performance of PRIME (as measured by median latency of the accelerator found across five runs) under various train-test splits on three applications studied in Table 30.

Applications	Best 5% Validation	Best 20% Validation (Table 30)	Worst 20% Validation
MobileNetEdgeTPU	273.30	298.50	286.53
M6	131.46	131.46	142.68
t-RNN Enc	137.45	130.67	135.71

H.2 DETAILS OF PRIME

In this section, we provide training details of our method PRIME including hyperparameters and compute requirements and details of different tasks.

H.2.1 *Hyperparameter and Training Details*

Algorithm 9 outlines our overall system for accelerator design. PRIME parameterizes the function $f_\theta(\boldsymbol{x})$ as a deep neural network as shown in Figure 13. The architecture of $f_\theta(\boldsymbol{x})$ first embeds the discrete-valued accelerator configuration $\boldsymbol{x}$ into a continuous-valued 640-dimensional embedding via two layers of a self-attention transformer [330]. Rather than directly converting this 640-dimensional embedding into a scalar output via a simple feed-forward network, which we found a bit unstable to train with Equation 10.4.2, possibly due to the presence of competing objectives for a comparison), we pass the 640-dimensional embedding into M different networks that map it to M different scalar predictions $(f^i_\theta(\boldsymbol{x}))_{i=1}^M$. Finally, akin to attention [330] and mixture of experts [294], we train an additional head to predict weights $(w_i)_{i=1}^M \geq 0$ of a linear combination of the predictions at different heads that would be equal to the final prediction: $f_\theta(\boldsymbol{x}) = \sum_{i=1}^K w_i f^i_\theta(\boldsymbol{x})$. Such an architecture allows the model to use different predictions $f^i_\theta(\boldsymbol{x})$, depending upon the input, which allows for more stable training. To train $f_\theta(\boldsymbol{x})$, we utilize the Adam [169] optimizer. Equation 10.4.2 utilizes a procedure Opt that maximizes the learned function approximately. We utilize the same technique as Section 10.4 ("optimizing the learned surrogate") to obtain these negative samples. We periodically refresh Opt, once in every 20K gradient steps on $f_\theta(\boldsymbol{x})$ over training.

The hyperparameters for training the conservative surrogate in Equations 10.4.2 and its contextual version are as follows:

- **Architecture of $f_\theta(\boldsymbol{x})$.** As indicated in Figure 13, our architecture takes in list of categorical (one-hot) values of different accelerator parameters (listed in Table 28), converts each parameter into 64-dimensional embedding, thus obtaining a 10×64 sized matrix for each accelerator, and then runs two layers of self-attention [330] on it. The resulting 10×64 output is flattened to a vector in $\mathbb{R}^{640}$ and fed into $M = 7$ different prediction networks that give rise to $f^1_\theta(\boldsymbol{x}), \cdots, f^M_\theta(\boldsymbol{x})$, and an additional attention 2-layer feed-forward network (layer sizes $= [256, 256]$) that determines

Algorithm 9 Training the conservative surrogate in PRIME

Initialize a neural model $f_{\theta_0}(\boldsymbol{x})$ and a set $M = 23$ of negative particles to be updated by the firefly optimizer $\{\boldsymbol{x}_1^-(0), \cdots, \boldsymbol{x}_i^-(0), \boldsymbol{x}_M^-(0)\}$ to random configurations from the design space.
for iteration $i = 0, 1, 2, 3, \ldots$ until convergence **do**
  for firefly update step $t = 0, 1, \ldots, 4$ ▷ **Inner loop do**
    Update the M fireflies according to the firefly update rule in Equation H.2.2,
    towards maximizing $f_{\theta_i}(\boldsymbol{x})$ according to: ▷ **Negative mining**
$$\boldsymbol{x}_i(ti+1) = \boldsymbol{x}_i(ti) + \beta(\boldsymbol{x}_i(ti) - \boldsymbol{x}_j(ti)) + \eta\epsilon_{ti}$$
  end for
  Find the best firefly found in these steps to be used as the negative sample:
$$\boldsymbol{x}_i^- = \arg\min\{f_{\theta_i}(\boldsymbol{x}_1^-(ti)), \cdots, f_{\theta_i}(\boldsymbol{x}_M^-(ti))\}$$ ▷ **Find negative sample**
  Run one gradient step on θ_i using Equation 10.4.2 with $\boldsymbol{x}_i^-$ as the negative sample
  if $i\%p == 0$, (p = 20000), then: ▷ **Periodically reinitialize the optimizer**
    Reinitialize firefly particles $\{\boldsymbol{x}_1^-(0), \cdots, \boldsymbol{x}_i^-(0), \boldsymbol{x}_M^-(0)\}$ to random designs.
end for
Return the final model $f_{\theta^*}(\boldsymbol{x})$

weights $w_1, \cdots, w_M$, such that $w_i \geq 0$ and $\sum_{i=1}^M w_i = 1$. Finally the output is simply $f_\theta(\boldsymbol{x}) = \sum_i w_i f_\theta^i(\boldsymbol{x})$.

- **Optimizer/learning rate for training** $f_\theta(\boldsymbol{x})$. Adam, $1e-4$, default $\beta_1 = 0.9$, $\beta_2 = 0.999$.
- **Validation set split.** Top 20% high scoring points in the training dataset are used to provide a validation set for deciding coefficients α, β and the checkpoint to evaluate.
- **Ranges of** α**,** β**.** We trained several $f_\theta(\boldsymbol{x})$ models with $\alpha \in [0.0, 0.01, 0.1, 0.5, 1.0, 5.0]$ and $\beta \in [0.0, 0.01, 5.0, 0.1, 1.0]$. Then we selected the best values of α and β based on the highest Kendall's ranking correlation on the validation set. Kendall's ranking correlation between two sets of objective values: $S = \{y_1, y_2, \cdots, y_N\}$ corresponding to ground truth latency values on the validation set and $S' = \{y'_1, y'_2, \cdots, y'_N\}$ corresponding to the predicted latency values on the validation set is given by τ equal to:
$$\tau = \frac{\sum_{i,j}^{N,N} \mathbb{I}[(y_i - y_j)(y'_i - y'_j) > 0] - \sum_{i,j}^{N,N} \mathbb{I}[(y_i - y_j)(y'_i - y'_j) \leq 0]}{N \cdot (N-1)}. \quad \text{(H.2.1)}$$
- **Clipping** $f_\theta(\boldsymbol{x})$ **during training**. Equation 10.4.2 increases the value of the learned function $f_\theta(\boldsymbol{x})$ at $\boldsymbol{x} = \boldsymbol{x}_0 \in \mathcal{D}_{\text{infeasible}}$ and $\boldsymbol{x}^- \sim \text{Opt}(f_\theta)$. We found that with the small dataset, these linear objectives can run into numerical instability, and produce $+\infty$ predictions. To avoid this, we clip the predicted function value both above and below by ± 10000.0, where the valid range of ground-truth values is $\mathcal{O}(1000)$.

- **Negative sampling with** $\mathrm{Opt}(\cdot)$. As discussed in Section 10.4, we utilize the firefly optimizer for both the negative sampling step and the final optimization of the learned conservative surrogate. When used during negative sampling, we refresh (i.e., reinitialize) the firefly parameters after every $p = 20000$ gradient steps of training the conservative surrogate, and run $t = 5$ steps of firefly optimization per gradient step taken on the conservative surrogate.
- **Details of firefly:** The initial population of fireflies depends on the number of accelerator configurations ($\mathcal{C}$) following the formula $10 + \lfloor (\mathcal{C}^{1.2} + \mathcal{C}) \times 0.5 \rceil$. In our setting with ten accelerator parameters (See Table 28), the initial population of fireflies is 23. We use the same hyperparameters: $\gamma = 1.0, \beta_0 = 1.0$, for the optimizer in all the experiments and never modify it. The update to a particular optimization particle (i.e., a firefly) $\boldsymbol{x}_i$, at the t-th step of optimization is given by:

$$\boldsymbol{x}_i(t+1) = \boldsymbol{x}_i(t) + \beta(\boldsymbol{x}_i(t) - \boldsymbol{x}_j(t)) + \text{ i.i.d. Gaussian noise}, \tag{H.2.2}$$

  where $\boldsymbol{x}_j(t), j \neq i$ is a different firefly that achieves a better objective value compared to $\boldsymbol{x}_i$ and the function β is given by: $\beta(r) = \beta_0 e^{-\gamma r^2}$.
- **Training set details:** The training dataset sizes for the studied applications are shown in Table 79. To recap, to generate the dataset, we first randomly sampled accelerators from the deign space, and evaluated them for the target application, and constituted the training set from the worst-performing feasible accelerators for the given application. Since different applications admit different feasibility criteria (differences in compilation, hardware realization, and etc.), the dataset sizes for each application are different, as the number of feasible points is different. Note however that as mentioned in the main text, these datasets all contain ≤ 8000 feasible points.

  Discussion on data quality: In the cases of t-RNN Dec, t-RNN Enc, and U-Net, we find that the number of feasible points is much smaller compared to other applications, and we suspect this is because our random sampling procedure does not find enough feasible points. This is a limitation of our data collection strategy and we intentionally chose this naïve strategy to keep data collection simple. Other techniques for improving data collection and making sure that the data does not consist of only infeasible points includes strategies such as utilizing logged data from past runs of online evolutionary methods, mixed with some data collected via random sampling to improve coverage of the design space.

H.2.1.1 *Details of Firefly Used for Our Online Evolutionary Method*

In this section, we discuss some details for firefly optimization used in the online evolutionary method.

Stopping criterion: We stopped the firefly optimization when the latency of the best design found did not improve over the previous 1000 iterations, but we also made sure

Table 79: Dataset sizes for various applications that we study in this paper. Observe that all of the datasets are smaller than 8000.

Application	Dataset size
MobileNetEdgeTPU	7697
MobileNetV2	7620
MobileNetV3	5687
M4	3763
M5	5735
M6	7529
U-Net	557
t-RNN Dec	1211
t-RNN Enc	1240

Table 80: Comparing the latency of the accelerators designed by the evolutionary approach for variable number of simulator access budgets (8k and 32k). Even with 4× as much allowed simulator interaction, online methods are unable to perform that well in our case.

Application	Area	PRIME	Evolutionary (Online)	
			8k data points	32k data points
MobileNetEdgeTPU	29 mm^2	298.50	320.28	311.35
t-RNN Dec	29 mm^2	132.88	770.93	770.63
t-RNN Enc	29 mm^2	130.67	865.07	865.07
Geomean of PRIME 's Improvement	—	(1.0×, 1.0×)	3.45×	3.42×

to run firefly optimization for at least 8000 iterations, to make sure that both the online and offline methods match in terms of the data budget. We also provide the convergence curves for firefly optimization on various single-application problems from Table 30 in Figure 53.

What happens if we run firefly optimization for longer? We also experimented with running the evolutionary methods for longer (i.e., 32k simulator accesses compared to 8k), to check if this improves the performance of the evolutionary approach. As shown in Table 80, we find that while this procedure does improve performance in some cases, the performance does not improve much beyond 8k steps. This indicates that there is a possibility that online methods can perform better than PRIME if they are run for many more optimization iterations against the simulator, but they may not be as data-efficient as PRIME .

Hyperparameter tuning for firefly: Since the online optimization algorithms we run have access to querying the simulator over the course of training, we can simply utilize the value of the latest proposed design as a way to perform early stopping and hyperparameter

tuning. A naïve way to perform hyperparameter tuning for such evolutionary methods is to run the algorithm for multiple rounds with multiple hyperparameters, however this is compute and time intensive. Therefore, we adopted a dynamic hyperparameter tuning strategy. Our implementation of the firefly optimizer tunes hyperparameters by scoring a set of hyperparameters based on its best performance over a sliding window of T data points. This allows us to adapt to the best hyperparameters on the fly, within the course of optimization, effectively balancing the number of runs that need to be run in the simulator and hyperparameter tuning. This dynamic hyperparameter tuning strategy requires some initial coverage of the hyperparameter space before hyperparameter tuning begins, and therefore, this tuning begins only after 750 datapoints. After this initial phase, every $T = 50$ iterations, the parameters γ and β_0 are updated via an evolutionary scoring strategy towards their best value.

Discussion of t-RNN Enc and t-RNN Dec. Finally, we discuss the results of the evolutionary approach on the t-RNN Enc and t-RNN Dec tasks, for which the convergence plots are shown in Figures 53h and 53i. Observe that the best solution found by this optimization procedure converges quite quickly in this case (with about 1000 iterations) and the evolutionary method, despite the dynamic hyperparameter tuning is unable to find a better solution. We hypothesize that this is because the performance of a local optimization method may suffer heavily due to the poor landscape of the objective function, and it may get stuck if it continuously observes only infeasible points over the course of optimization.

H.2.1.2 *Exact Hyperparameters Found By Our Cross-Validation Strategy*

In this section, we present the exact hyperparameters found by our cross-validation strategy discussed in Section 10.4. To recap, our offline cross-validation strategy finds the early stopping checkpoint and selects the values of α and β in Equation 10.4.2 that attain the highest rank correlation on a held-out validation set consisting of top 20% of the dataset feasible samples. The values of α, β and checkpoint selected for the experiments in Table 3, Table 4, Table 5 and 6 are shown in Table 81.

H.2.2 *Details of Architecting Accelerators for Multiple Applications Simultaneously*

Now we will provide details of the tasks from Table 31 where the goal is to architect an accelerator which is jointly optimized for multiple application models. For such tasks, we augment data-points for each model with the context vector c_k from Table 29 that summarizes certain parameters for each application. For entries in this context vector that have extremely high magnitudes (e.g., model parameters and number of compute operations), we normalize the values by the sum of values across the applications considered to only encode the relative scale, and not the absolute value which is not required. To better visualize the number of feasible accelerators for joint optimization,

Table 81: Hyperparameters α, β and checkpoint index (measured in terms of gradient steps on the learned conservative model) for PRIME found by our **offline** cross-validation strategy discussed in Section 10.4, that is based on the Kendall's rank correlation on the validation set (note that no simulator queries were used to tune hyperparameters). In the case of the multi-task and zero-shot scenarios, when training on more than one application, the batch size used for training PRIME increases to N-fold, where N is the number of applications in the training set, therefore we likely find that even a few gradient steps are good enough.

Table	Application	α	β	Checkpoint Index
Table 30	MobileNetEdgeTPU	0.01	5.0	80000
Table 30	MobileNetV2	5.0	5.0	120000
Table 30	MobileNetV3	5.0	0.01	80000
Table 30	M4	0.1	0.0	80000
Table 30	M5	5.0	1.0	80000
Table 30	M6	1.0	1.0	60000
Table 30	U-Net	0.0	1.0	100000
Table 30	t-RNN Dec	1.0	0.0	60000
Table 30	t-RNN Enc	0.0	0.1	60000
Table 31	MobileNet (EdgeTPU, V2, V3)	5.0	0.01	60000
Table 31	MobileNet (V2, V3), M5, M6	0.0	5.0	30000
Table 31	MobileNet (EdgeTPU, V2, V3), M4, M5, M6	0.5	0.0	100000
Table 31	MobileNet (EdgeTPU, V2, V3), M4, M5, M6, t-RNN Enc (Area 29.0)	0.0	1.0	20000
Table 31	MobileNet (EdgeTPU, V2, V3), M4, M5, M6, t-RNN Enc (Area 100.0)	0.0	0.0	20000
Table 31	MobileNet (EdgeTPU, V2, V3), M4, M5, M6, U-Net, t-RNN (Enc, Dec) (Area 29.0)	0.01	0.01	10000
Table 31	MobileNet (EdgeTPU, V2, V3), M4, M5, M6, U-Net, t-RNN (Enc, Dec) (Area 100.0)	0.01	0.1	20000
Table 32	Train (Zero-Shot): MobileNet (EdgeTPU, V3)	5.0	0.01	60000
Table 32	Train (Zero-Shot): MobileNet (V2, V3), M5, M6	0.0	5.0	30000
Table 32	Train (Zero-Shot): MobileNet (EdgeTPU, V2, V3), M4, M5, M6, t-RNN Enc (Area 29.0)	0.0	1.0	20000
Table 32	Train (Zero-Shot): MobileNet (EdgeTPU, V2, V3), M4, M5, M6, t-RNN Enc (Area 100.0)	0.1	5.0	20000
Table 33	MobileNetV2 (NVDLA)	0.0	1.0	40000
Table 33	MobileNetV2 (ShinDianNao)	0.0	0.0	40000
Table 33	ResNet 50 (NVDLA)	0.01	0.0	40000
Table 33	ResNet 50 (ShinDianNao)	0.0	0.0	75000
Table 33	Transformer (NVDLA)	0.01	1.0	200000
Table 33	Transformer (ShinDianNao)	0.0	0.1	100000

Figure 54 show the tSNE plot (raw architecture configurations are used as input) of high-performing accelerator configurations. The blue-colored dots are the jointly feasible accelerators in the combined dataset, and note that these data points are no more than 20-30 in total. The highlighted red star presents the best design suggested by PRIME with average latency of 334.70 (Table 31). This indicates that this contextual, multi-application problem poses a challenge for data-driven methods: these methods need to produce optimized designs even though very few accelerators are jointly feasible in the combined dataset. Despite this limitation, PRIME successfully finds more efficient accelerator configurations that attain low latency values on each of the applications jointly, as shown in Table 31.

H.2.3 *Dataset Sensitivity to Accelerator Parameters*

We visualize the sensitivity of the objective function (e.g. latency) with respect to the changes in certain accelerator parameters, such as memory size (Table 28), in Figure 56b, illustrating this sensitivity. As shown in the Figure, the latency objective that we seek to optimize can exhibit high sensitivity to small variations in the architecture parameters, making the optimization landscape particularly ill-behaved. Thus, a small change in one of the discrete parameters, can induce a large change in the optimization objective. This characteristic of the dataset further makes the optimization task challenging.

H.3 Overview of Accelerators and Search Space

This section briefly discuss the additional accelerators (similar to [164]) that we evaluate in this work, namely NVDLA [250] and ShiDianNao [67], and their corresponding search spaces.

NVDLA: Nvidia Deep Learning Accelerator NVDLA [249] is an open architecture inference accelerator designed and maintained by Nvidia. In compared to other inference accelerators, NVDLA is a weight stationary accelerator. That is, it retains the model parameters on each processing elements and parallelizes the computations across input and output channels. NVDLA-style dataflow accelerators generally yield better performance for the computations of layers at the later processing stages. This is because these layers generally have larger model parameters that could benefit from less data movement associated to the model parameters.

ShiDianNao: Vision Accelerator Figure 55 shows the high-level schematic of ShiDianNao accelerator [67]. ShiDianNao-style dataflow accelerator is an output-stationary accelerator. That is, it keeps the partial results inside each PE and instead move the model parameters and input channel data. As such, in compared to NVDLA-style accelerators, ShiDianNao provides better performance for the computations of the layers with large output channels (generally first few layers of a model).

Search space of dataflow accelerators. We follow a similar methodology as [164] to evaluate additional hardware accelerators, discussed in the previous paragraphs. We use MAESTRO [190], an analytical cost model, that supports the performance modeling of various dataflow accelerators. In this joint accelerator design and dataflow optimization problem, the total number of parameters to be optimized is up to 106—the tuple of (# of PEs, Buffers) per per model layer—with each parameter taking one of 12 discrete values. This makes the hardware search space consist of $\approx 2.5 \times 10^{114}$ accelerator configurations. We also note that while the method proposed in [164] treats the accelerator design problem as a sequential decision making problem, and uses reinforcement learning techniques, PRIME simply designs the whole accelerator in a single step, treating it as a model-based optimization problem.

Table 82: The evaluated applications, their model parameter size, number of compute operations, and normalized compute-to-memory ratio.

Name	Model Param	# of Compute Ops.	Normalized Compute-to-Memory Ratio
MobileNetEdgeTPU	3.87 MB	1,989,811,168	1.38e-1
MobileNetV2	3.31 MB	609,353,376	4.96e-2
MobileNetV3	5.20 MB	449,219,600	2.33e-2
M4	6.23 MB	3,471,920,128	1.5e-1
M5	2.16 MB	939,752,960	1.17e-1
M6	0.41 MB	228,146,848	1.5e-1
U-Net	3.69 MB	13,707,214,848	1.0
t-RNN Dec	19 MB	40,116,224	**5.68e-4**
t-RNN Enc	21.62 MB	45,621,248	**5.68e-4**

Table 83: Additional ablation study under zero-shot setting when the test applications include all the nine evaluated models (e.g. MobileNet (EdgeTPU, V2, V3), M4, M5, M6, t-RNN Dec, t-RNN Enc, U-Net). Lower latency is better. From **left** to **right**: the applications used to train the surrogate model in PRIME , the area constraint of the accelerator, PRIME 's (best, median) latency.

Train Applications	Area	PRIME (best, median)
(1) MobileNet (V2, V3), M5, M6	29 mm^2	(426.65, 427.35)
(2) MobileNet (V2, V3), M5, t-RNN Enc	29 mm^2	(461.79, 464.87)
(3) MobileNet (EdgeTPU, V2, V3), M4, M5, M6, t-RNN Enc	29 mm^2	(426.65, 427.94)

H.4 SUBSET OF APPLICATIONS FOR GOOD ZERO-SHOT PERFORMANCE

In this section, we present the results of an ablation study with the goal to identify a subset of applications such that training on data from only these applications yields good zero-shot performance across all the nine applications studied in this work. Since we cannot train PRIME for every subset of applications because the space of subsets of all applications is exponentially large, we utilized some heuristics in devising the subset of applications we would train on, with the goal to make interesting observations that allow us to devise rough guidelines for performing application selection.

Our heuristic for devising subsets of applications: Building on the intuition that applications with very different compute to memory ratios (shown in Table 82) may require different accelerator designs – for example, if our goal is to run a compute-intensive application, we likely need an accelerator design with more compute units – we study two subsets of training applications: **(1)** MobileNetV2, MobileNetV3, M6, M5, and **(2)** MobileNetV2, MobilenetV3, M5, t-RNN Enc. Note that, these two combinations only differ in whether some RNN application was used in training or not. As shown in Table 82, the t-RNN applications admit a very different compute to memory ratio, for instance, while this ratio is $5.68e-4$ for t-RNN Enc and t-RNN Dec, it is much different $\sim 0.01 - 0.2$ for other models MobileNetEdgeTPU, MobileNetV2, MobileNetV3, M5, and M6. This means that likely t-RNN Enc and Dec will require different kinds of accelerators for good performance compared to the other applications.

Results: We present the performance of zero-shot evaluating the designed accelerator obtained by training on combinations **(1)** and **(2)**, and also the accelerator obtained by training on **(3)** seven applications from Table 32 in Table 83 as reference. We make some key takeaways from the results:

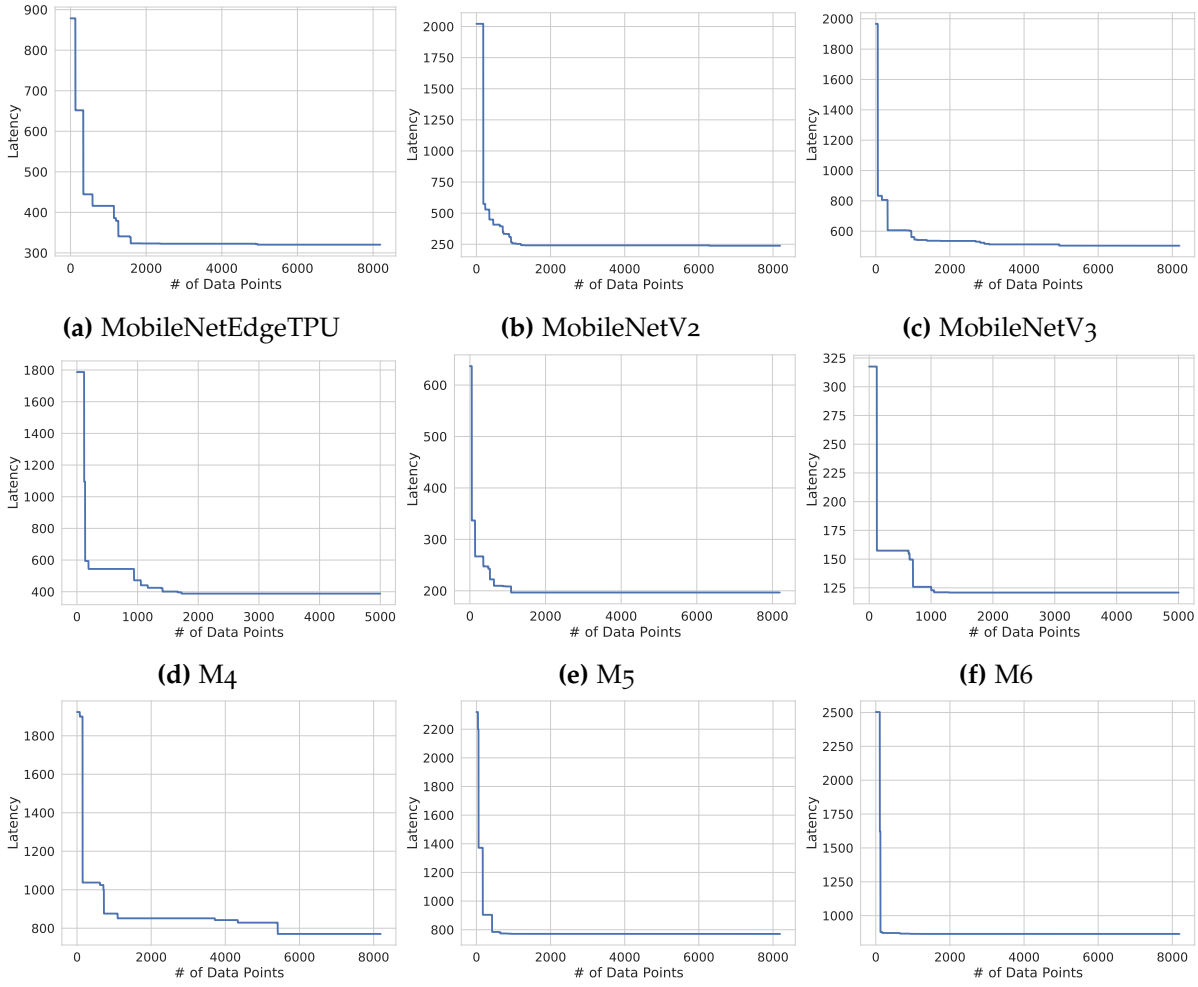
- The performance of both configuration **(1)** and training with seven applications (**(3)**, last row of Table 83) are similar.
- In case **(2)**, when the training applications consist of four applications in which one application is t-RNN Enc, with drastically different compute to memory ratio (Table 82), the performance on an average across all applications becomes slightly worse (compare the performance in **(2)** vs **(3)**).

Conclusion and guidance on selecting good applications: The above results indicate that only a few applications (e.g., four applications in case **(1)**) can be enough for good performance on all nine applications. While this set may not be not minimal, it is certainly a much smaller set compared to the nine applications considered. Adding an RNN application in case **(2)** increases latency in compared to case **(1)**, because t-RNN Enc likely admits a very different optimal accelerator compared to the other applications due to a very different compute/memory ratio, which in turn skews the generalization of the surrogate learned by PRIME when trained only on this limited set of four applications. However, when seven applications are provided in case **(3)**, even when the set of training applications includes t-RNN, its contribution on the PRIME surrogate is reduced since many other compute intensive applications are also provided in the training set and the resulting accelerator performs well.

Practitioner guidance: The primary practitioner guidance we can conclude here is that *the models used for training must be representative of the overall distribution of the target models that we want to zero-shot generalize to*. Since a number of our applications are compute intensive, we were able to simply utilize set **(1)** to attain good performance on all the applications. On the other hand, in case **(2)**, when the t-RNN Enc application was over-represented – while seven of nine applications we considered were primarily compute intensive, one out of four applications we used for training in case **(2)** were memory intensive – this hurt performance on the overall set. Therefore, we believe that ensuring that the training subset of applications is adequately aligned with the overall set of applications in terms of compute/memory ratio statistic is imperative for favorable zero-shot performance.

For a practitioner deciding between zero-shot generalization and additional data collection, it may make sense to test if the target application admits a similar value of the compute to memory ratio as an already existing application. If it does, then the practitioner might be able to utilize the zero-shot generalization, as is indicated with the good performance of case **(1)**, whereas if the compute/memory ratio is heavily different from any seen application, zero-shot generalization to the target application

may be worse. Finally, making sure that the training applications adequately reflect the compute/memory ratio statistic for the overall target set is important.



(a) MobileNetEdgeTPU **(b)** MobileNetV2 **(c)** MobileNetV3

(d) M4 **(e)** M5 **(f)** M6

(g) U-Net **(h)** t-RNN Dec **(i)** t-RNN Enc

Figure 53: Optimization behavior of Firefly optimizer (Online Evolutionary). Observe that the optimization procedure converges and plateaus very quickly (at least 1000 iterations in advance) and hence we stop at 8000 iterations. In the case of t-RNN Enc and t-RNN Dec, we find that the evolutionary algorithm performs poorly and we suspect this is because it saturates quite quickly to a suboptimal solution and is unable to escape. This is also evident from Figures 53h and 53i, where we observe that online optimization plateaus the fastest for these RNN applications.

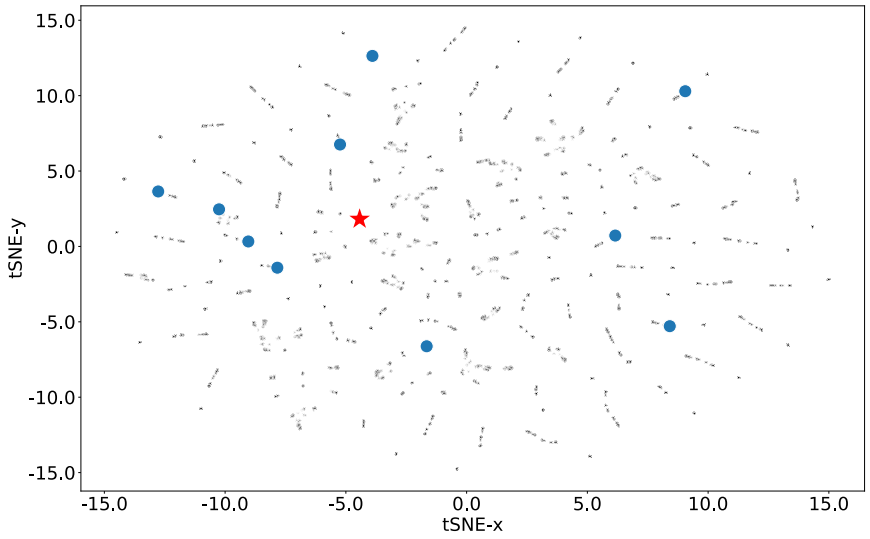


Figure 54: tSNE plot of the joint dataset and randomly sampled infeasible data points. The blue points show the accelerator configurations that are jointly feasible for all the applications. The highlighted point with red star shows the best design proposed by PRIME Ṫhe rest of the points show the infeasible points.

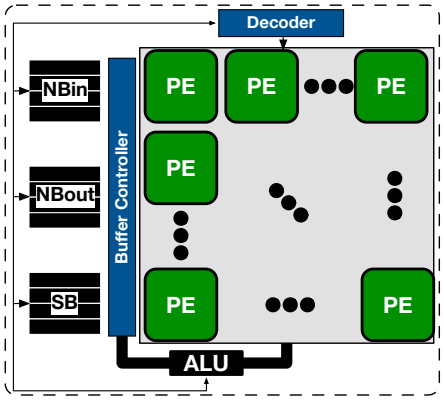
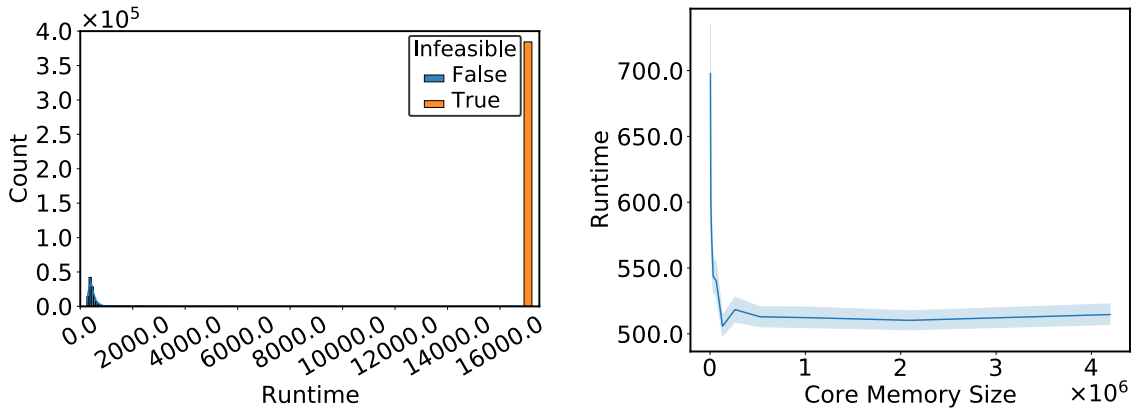


Figure 55: Overview of ShiDianNao dataflow accelerator. This dataflow accelerators exhibits an output-stationary dataflow where it keeps the partial results stationary within each processing elements (PEs).



(a) **(b)**

Figure 56: The (a) histogram of infeasible (orange bar with large score values)/feasible (blue bars) data points and (b) the sensitivity of runtime to the size of core memory for the MobileNetEdgeTPU [140] dataset.

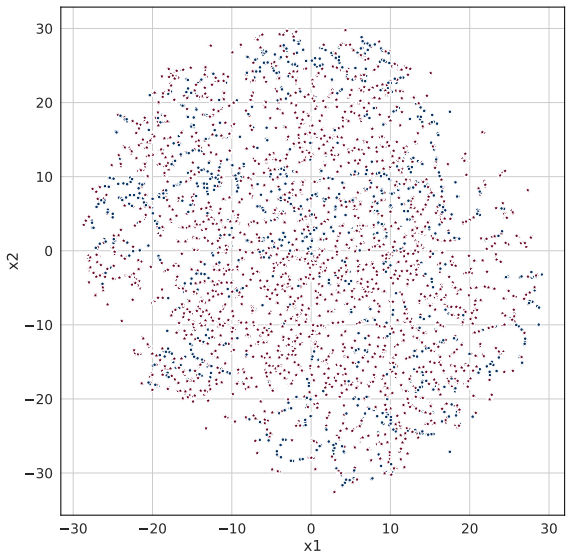


Figure 57: tSNE plot of the infeasible and feasible hardware accelerator designs. Note that feasible designs (shown in blue) are embedded in a sea of infeasible designs (shown in red), which makes this a challenging domain for optimization methods.

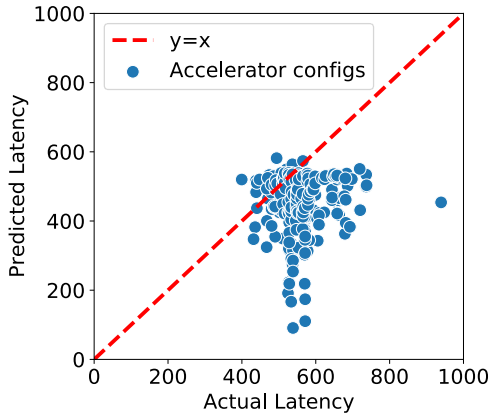


Figure 58: To verify if the overestimation hypothesis–that optimizing an accelerator against a naïve standard surrogate model is likely to find optimizers that appear promising under the learned model, but do not actually attain low-latency values–in our domain, we plot a calibration plot of the top accelerator designs found by optimizing a naïvely trained standard surrogate model. In the scatter plot, we represent each accelerator as a point with its x-coordinate equal to the actual latency obtained by running this accelerator design in the simulator and the y-coordinate equal to the predicted latency under the learned surrogate. Note that for a large chunk of designs, the predicted latency is much smaller than their actual latency (i.e., these designs lie beneath the $y = x$ line in the plot above). This means that optimizing designs under a naïve surrogate model is prone to finding designs that appear overly promising (i.e., attain lower predicted latency values), but are not actually promising. This confirms the presence of the overestimation hypothesis on our problem domain.

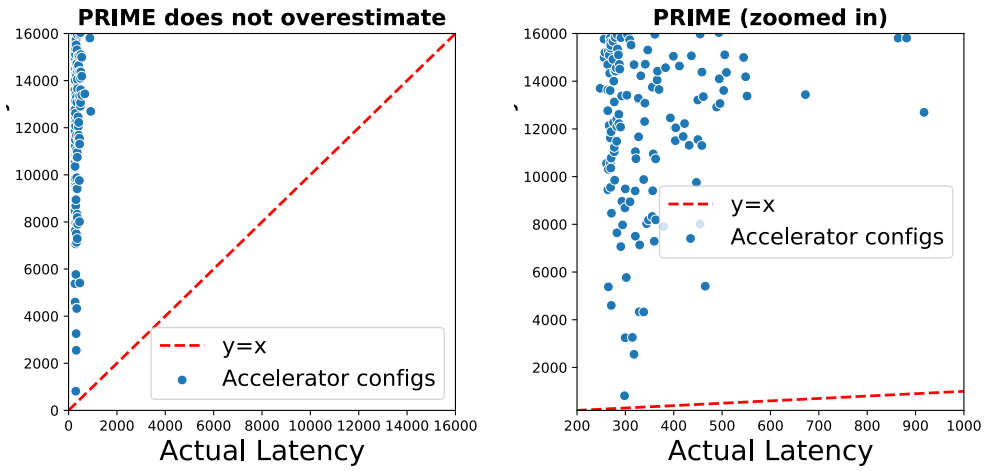


Figure 59: Plot showing the calibration plot of the predicted (y-axis) and actual latencies (x-axis) of accelerators found by PRIME . Compared to Figure 58, observe that all the acclerator configurations lie above $y = x$, meaning that PRIME predicts a higher latency (y-axis) compared to the actual latency. This means that PRIME does not think that accelerators that attain high-latency values under the simulator, are actually good. We also provide a zoomed-in version of the plot on the right, which shows that there are accelerators do have meaningfully distinct latency predictions under PRIME . Observe in the zoomed-in plot that the designs that attain small predicted latencies also perform relatively better under the actual latency compared to the designs that attain larger predicted latency of $\sim$ 14000-16000 under the PRIME surrogate. Optimizing against PRIME is still effective because optimization just needs relative correctness of values, not absolutely correct latency predictions.