On the Effectiveness of Fine Tuning versus Meta-reinforcement Learning

Mandi Zhao
Pieter Abbeel
Stephen James

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2022-202
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2022/EECS-2022-202.html

August 12, 2022

Acknowledgement

This work was supported by DARPA RACER and Hong Kong Centre for Logistics Robotics. The authors would like to thank Shikun Liu, Danijar Hafnar, Olivia Watkins, Yuqing Du, and Luisa Zintgraf for their help and feedback.

On the Effectiveness of Fine Tuning versus Meta-reinforcement Learning

by Author Mandi Zhao

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Professor Pieter Abbeel
Research Advisor

8/10/2022

(Date)

* * * * * * *

Professor Ken Goldberg
Second Reader

8-12-2022

(Date)

On the Effectiveness of Fine-tuning Versus Meta-reinforcement Learning

Abstract

Intelligent agents should have the ability to leverage knowledge from previously learned tasks in order to learn new ones quickly and efficiently. Meta-learning approaches have emerged as a popular solution to achieve this. However, meta-reinforcement learning (meta-RL) algorithms have thus far been restricted to simple environments with narrow task distributions. Moreover, the paradigm of pretraining followed by fine-tuning to adapt to new tasks has emerged as a simple yet effective solution in supervised and self-supervised learning. This calls into question the benefits of meta-learning approaches in reinforcement learning, which typically come at the cost of high complexity. We therefore investigate meta-RL approaches in a variety of vision-based benchmarks, including Procgen, RLBench, and Atari, where evaluations are made on completely novel tasks. Our findings show that when meta-learning approaches are evaluated on different tasks (rather than different variations of the same task), multi-task pretraining with fine-tuning on new tasks performs equally as well, or better, than meta-pretraining with meta test-time adaptation. This is encouraging for future research, as multi-task pretraining tends to be simpler and computationally cheaper than meta-RL. From these findings, we advocate for evaluating future meta-RL methods on more challenging tasks and including multi-task pretraining with fine-tuning as a simple, yet strong baseline. [1]

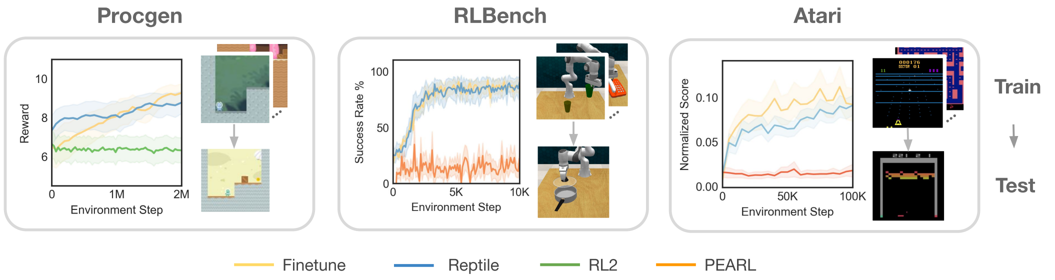


Figure 1: Meta reinforcement learning (meta-RL) algorithms have thus far been restricted to simple environments with narrow task distributions. Our study investigates meta-RL algorithms on wide task distributions across 3 benchmarks, and concludes that **multi-task pretraining with fine-tuning on new tasks performs equally as well, or better, than meta-RL.** Plots show the average performance across all tasks in each of the 3 benchmarks.

1 Introduction

One of the major gaps between human and machine intelligence is the sample efficiency of learning. Whereas humans can leverage past knowledge to learn a new task from a few examples, current

[1] Project Website: https://sites.google.com/berkeley.edu/finetune-vs-metarl

machine learning systems often require a large amount of data and supervision to achieve even a single task. To bridge this gap, meta-learning has become a popular approach — it uses many tasks to meta-train an optimal learning strategy, which enables few-shot generalization on a test task.

Meta-learning methods have had the most success in supervised learning settings [1–3], specifically few-shot image classification, where the goal is to learn a classifier to recognize unseen classes during a test-time training phase with limited labeled data. Recent work has found that variations of simple pretraining and fine-tuning can perform equally as well as more complex meta-learning approaches [4–7]. However, within the realm of reinforcement learning, simple pretraining and fine-tuning is not known to out perform meta-reinforcement learning (meta-RL). Our hypothesis for this discrepancy is simple: the computer vision (CV) community evaluates their approaches on distinct test tasks (e.g. classifying dogs, cats, and birds), while the meta-RL community evaluates on *variations* of the same train-time tasks; for example, varying transition dynamics (e.g. different friction parameters) or varying reward functions (e.g. running forward v.s. running backward) are better categorized as variations rather than different tasks, as discussed in recent work [8, 9]. *Variation* adaptation is inherently easier than *task* adaptation, and does not paint a full picture of the shortcomings of meta-RL. Moreover, most meta-RL methods (with a few exceptions, discussed in 6) have been studied in fully observable settings or with shaped rewards [10–12], neglecting more realistic real-world scenarios, where rewards are often sparse, and observations are high-dimensional (e.g. images, point-clouds, etc).

Evidently, there is a gap in the literature, where a large-scale study would be well placed to analyze the setting of vision-based meta-RL across a *truly* diverse set of tasks. We use 3 existing RL benchmarks: Procgen [13], RLBench [8], and Atari Learning Environment (ALE) [14, 15] — each offers a diverse set of distinct tasks that we use for train and test. For example in RLBench [8], an agent could be trained to pick up cups, take a USB out of a computer, and reach target locations, while at test time, adaptation would be evaluated on completely unseen tasks, such as lifting blocks and pushing buttons.

We investigate three prominent meta-RL algorithms of differing paradigms: Reptile [11] — a gradient-based method, PEARL [12] — a context-based method, and RL^2 [16] — an LSTM-based method. Results from this study are enlightening: multi-task pretraining, followed by fine-tuning on novel tasks, performs equally as well, or better, than all three meta-RL algorithms, while being much simpler and less computationally expensive to train. In light of this, we advocate for future research to shift towards more challenging benchmarks, and include multi-task pretraining with fine-tuning as a simple, yet strong baseline.

To summarize, the key **contributions** of our study are as follows:

- We show that multi-task pretraining followed by fine-tuning on novel tasks performs equally as well, or better, than common meta-RL baselines on vision-based environments.
- We present the first large-scale multi-task and meta-RL study on three existing benchmarks: cross-level adaptation on Procgen, cross-task adaptation on RLBench, and cross-game adaptation on Atari.
- Within RLBench, we show that large-scale multi-task pretraining can overcome sparse rewards on unseen test tasks and perform significantly better than training from scratch .

2 Preliminaries

Reinforcement Learning. Reinforcement learning (RL) assumes access to an agent that interacts with an environment in which there are states $\mathbf{s} \in \mathcal{S}$, actions $\mathbf{a} \in \mathcal{A}$, and a reward function $R(\mathbf{s}_t, \mathbf{a}_t)$, where t represents the current time step. The agent must discover a policy π that maximizes the expectation of the sum of discounted rewards, i.e., $\mathbb{E}_\pi[\sum_t \gamma^t R(\mathbf{s}_t, \mathbf{a}_t)]$, where future rewards are discounted by a factor $\gamma \in [0, 1)$. Each policy π has a corresponding Q-value function $Q(\mathbf{s}, \mathbf{a})$, which defines the agents expected return when following the policy after taking action $\mathbf{a}$ in state $\mathbf{s}$.

Meta-Reinforcement Learning. Meta-reinforcement learning (meta-RL) aims to quickly adapt to any task via RL. Given a distribution of tasks $p(\mathcal{T})$, where each sampled task $\mathcal{T}_i \in p(\mathcal{T})$, is a stand-alone MDP $\mathcal{T}_i = (\mathcal{S}, \mathcal{A}, P, \gamma, R)_i$, or partially observable MDP (POMDP) $\mathcal{T}_i = (\mathcal{S}, \mathcal{A}, P, \gamma, R, \mathcal{O})_i$, in the case where an agent receives observations $\mathbf{o} \in \mathcal{O}$ rather than states. Given a task $\mathcal{T}$, the agent is allowed to collect a small amount of data $\mathcal{D}_\mathcal{T}$ and use it to adapt the policy to obtain $\pi_\mathcal{T}$ with an RL algorithm. Meta-RL training aims to find an optimal initial policy or exploration strategy that maximizes the expected return of its adapted policies across all tasks within a limited number of steps: $\mathbb{E}_{\mathcal{T}\sim p(\mathcal{T}),(\mathbf{s}_t,\mathbf{a}_t)\sim\pi_{\mathcal{T}}}[\sum_t \gamma^t R(\mathbf{s}_t,\mathbf{a}_t)]$.

Assumption on Task Distribution. We assume no ambiguity between any two tasks in the train or test sets. Namely, all tasks have distinct, non-overlapping state-action spaces: $\forall \mathcal{T}_i, \mathcal{T}_j \in p(\mathcal{T}), i \neq j$, $\mathcal{S}_i \times \mathcal{A}_i \cap \mathcal{S}_j \times \mathcal{A}_j = \varnothing$. An agent should be able to infer which task to perform based on a single-step observation (e.g. whether it's playing Breakout or Pong). As a result, our multi-task pretraining baseline does not need a separate indicator to determine which task to complete. Note how this assumption is different from most prior work in meta-RL, which often assume the difference in "tasks" only occurs in transition dynamics (e.g. different friction parameters) or the reward function (e.g. running forward v.s. running backward) — hence we categorize those prior assumptions as "variation adaptation" to better distinguish from our work, as illustrated in Figure 2.

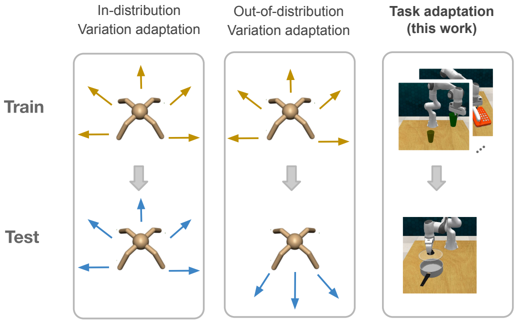


Figure 2: An illustrative comparison between the task assumptions from prior works' (in-distribution or out-of-distribution adaptation) and from ours (task adaptation). In our assumption, we emphasize both the high diversity of training tasks — which should lead to generalizable skills, and strictly unseen test-time tasks — which poses a more challenging adaptation problem than simple retrieval of trained skills.

2.1 Overview of Compared Algorithms

Reptile [11] is a first-order gradient-based meta-learning algorithm. It finds initial model parameters that can be quickly adapted to any task via gradient descent. Meta-training alternates between an inner loop — which performs multiple gradient steps on a single task, and an outer loop — which mixes the updated parameters from multiple tasks. Reptile is mathematically similar to first-order model-agnostic meta-learning [3] (MAML) while achieving comparable results as MAML in both supervised [11] and imitation learning [17] settings. We choose Reptile for its simplicity and applicability to both on-policy and off-policy RL algorithms.

PEARL [12] is a context-based, off-policy meta-RL algorithm. It meta-trains a policy that conditions on context variables, which are encoded from small amounts of an agent's experiences and are used to perform online probabilistic filtering to infer how to solve a new task. At test time, it adapts to a new task by collecting new experiences to update the context and adjusting its policy according to the new context.

RL^2 [16] is a recurrence-based, policy-gradient algorithm. It represents the RL policy with a recurrent neural network (RNN) that embeds agent experiences from multiple "trials" in a single task, with the aim of encoding an RL update rule into the RNN weights. At test time, it adapts by rolling out an agent in a new task, which updates the RNN hidden states and improves the policy to success after a few trials.

Multi-task Training As a baseline, we run multi-task reinforcement learning (MTRL), which simply learns to complete all the training tasks simultaneously with a single RL agent. At test time, the agent adapts to a new task by finetuning with the same RL algorithm as training.

Base RL Algorithms Each of the above methods is applied to a base RL algorithm that are different across benchmarks. For our Progen experiments (Section 3), we use PPO [18]. For our RLBench experiments (Section 5), we use C2F-ARM [19]. For our Atari experiments (Section 7), we use RainbowDQN [20] as the base algorithm. Justification for each of these choices is given in their respective sections. See Algorithms 1 and 2 in Appendix B for a unified overview of how the meta-RL algorithms are implemented.

All experiments in the study are trained on a maximum of 8 RTX A5000 GPUs, each with 24GB of memory. Unless otherwise indicated, we use the same hyperparameters and architectures as used in prior work; full details are given in the appendix.

3 Procgen Experiments

Procgen Benchmark [13] is a suite of procedurally-generated game-like environments designed for studying generalization in deep RL, where an agent is trained on a set of levels and evaluated on unseen levels. We classify Procgen as the easiest benchmark in this study, because both the embodiment and task objects are consistent across tasks, while only the colors and layouts vary.

Prior work has explored improving generalization by training on a large amount of levels [13], adding data-augmentation to visual inputs [21, 22], knowledge transfer during training [23], and self-supervised world models [24]. Beyond zero-shot generalization, few prior works use Procgen to evaluate meta-RL, with one exception being *Alver et al.* [25], which showed that RL^2 failed to generalize on simplified Procgen games.

We choose one representative Procgen game, namely Coinrun, where an agent is tasked to avoid obstacles and reach a golden coin. The game contains multiple "levels", each of which has a unique color theme or layout. We use up to 10,000 levels for training and held-out 20 levels for testing. 100 million environment steps are used for all pretraining runs, and we take the model checkpoint after 1500 PPO iterations for testing adaptation — the environment is run in parallel on 256 threads and each PPO iteration uses 256 steps.

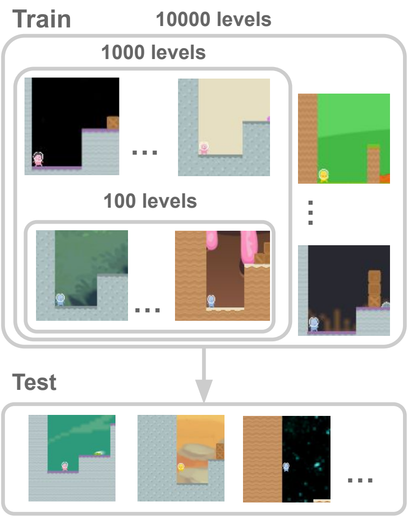


Figure 3: In Procgen experiments, each agent is trained on various number of Coinrun levels and tested on 20 unseen levels.

3.1 Training Setup

We use PPO [18] as the base RL algorithm and use the same image encoder as IMPALA [26], which has been shown to achieve competitive performance on single-game settings in Procgen [13, 27]. Note that PEARL is excluded from this section because it is off-policy.

Reptile-PPO incorporates the inner-outer loop in Reptile [11] with PPO. For each iteration in the inner loop, the environment is fixed to one randomly sampled task level, where rollouts are collected and used to perform 3 iterations of batched gradient updates. Following that, the outer loop takes the updated policy parameters, and performs a soft parameter update. This process is repeated iteratively.

RL^2-PPO combines an LSTM with PPO as proposed in [16]. We modify the Coinrun environment to re-sample the same level for multiple episodes, such that one original environment episode is treated as one "trial", and each RL^2 episode contains multiple "trials". Until an episode is done, the agent's past rollouts are concatenated and fed into an LSTM layer. The image observations are first encoded by the IMPALA (following [13]) and flattened into 256-dimensional latent vectors, then concatenated with action, reward, and done signals. The concatenated vectors are embedded again before fed into LSTM hidden layers.

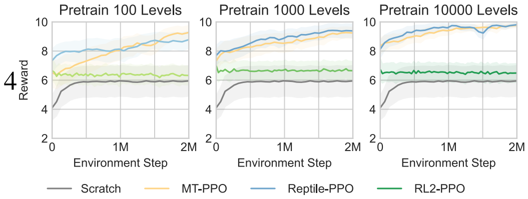
MT-PPO jointly trains PPO on multiple levels of Coinrun. The environment randomly samples from a fixed number of training levels, and the policy is updated with rollouts from multiple levels.

3.2 Test-time Adaptation Setup

For **Reptile-PPO** and **MT-PPO**, a trained agent is finetuned with vanilla PPO for 2 million environment steps on each test level. For **RL^2-PPO**, the LSTM hidden states are reset, and the agent is rolled-out on each new level for 2 million environment steps without gradient updates. We also compare adaptation with training from scratch, where one agent is trained for each test level with PPO for 2 million steps. We use a total of 20 held-out levels for testing, and results are reported by averaging across all 20 levels.

3.3 Results

Results for testing on unseen levels are shown in Figure 4. We first remark that **finetuning**



achieves the best performance at adaptation to new levels both in terms of sample efficiency and final performance, and is shown consistently across varying number of training levels. Notably, Reptile-PPO is also able to improve performance on test tasks, but the finetuning performance is worse than MT-PPO, which suggests the Reptile-learned parameters provides a less adaptable initialization than the simpler multi-task trained parameters. Figure 4 also shows that increasing the number of training levels improves zero-shot performance on MT-PPO (and less so for Reptile-PPO), which is consistent with the aforementioned prior work that studys generalization on Procgen. Expanding the training set also benefits finetuning and provides a clear advantage over training from scratch, which is encouraging news for simple scaling of training tasks as an alternative to designing computationally complex meta-training algorithms.

RL^2 does not improve performance on unseen levels, meaning that the updated hidden state of the RNN fails to adapt sufficiently while testing on new levels. This is consistent with [25], where RL^2 fails to adapt even after simplifying the level such that the coin is visible at the beginning of each trial.

4 RLBench Experiments

RLBench is a vision-based manipulation benchmark and learning environment, with a focus on sparse rewards, multi-task learning, and meta-learning. The environment has a rich set of more than 100 real-world inspired tasks involving diverse objects. The environment also provides easy access to expert demonstrations for all tasks, which is vital for overcoming the exploration problem imposed by the benchmark's sparse rewards. We classify RLBench as the medium difficulty benchmark in this study; while the embodiment is consistent across tasks (i.e., the Franka Panda robot arm), the task objects and goals vary drastically.

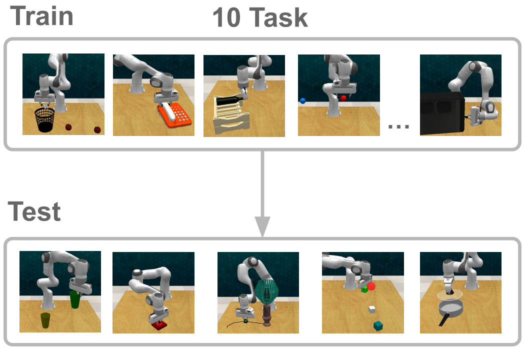


Figure 5: In RLBench experiments, each agent is trained on 10 tasks and evaluated on 5 unseen tasks.

To ensure the experiment results do not get affected by arbitrary task selection, we design a comprehensive set of train-test task splits that resemble cross-validation in the supervised learning setting. Specifically, we use a fixed set of 11 RLBench tasks and create 5 splits. Each split uses a (randomly selected) held-out task and trains an agent on the remaining 10 tasks.

4.1 Training Setup

We use C2F-ARM [19] as the base off-policy RL algorithm. This was chosen because more widely-used RL algorithm, such as DDPG [28], TD3 [29], SAC [30], and DrQ [22] are known to fail [31] in RLBench due to the challenging setup. C2F-ARM [19] is a vision-based robot manipulation algorithm that can learn sparse-reward reinforcement learning tasks by using a small number of initial demonstrations. C2F-ARM is described in more detail in Appendix D. Note that RL^2 is excluded from this section because it is on-policy.

Reptile-C2F-ARM modifies the off-policy batch update in C2F-ARM with an inner- and outer-loop proposed in Reptile [11]. At the beginning of training, each task is given a separate replay buffer, which is initialized with transitions collected from 5 demonstration trajectories and continuously appended with the agent's online experiences. During training, for multiple steps in the inner loop, the agent draws a batch from the replay buffer of a randomly sampled task and performs updates to the Q-attention. In the outer loop, the network gets a soft update to mix the parameters from before and after the inner loop updates.

PEARL-C2F-ARM conditions a context embedding to the Q-attention network. To obtain the context for a task, a batch of transitions is drawn from a window of recent agent experiences, and a separate convolution encoder is used to first encode the image observations individually. Then, each image embedding is concatenated with the action and reward, and together encoded into a single vector. Finally, the context embeddings are sampled as proposed in [12]. The context encoder is additionally trained with a KL loss.

MT-C2F-ARM jointly trains C2F-ARM on all training tasks. During each replay batch update, both MT-C2F-ARM draw samples from multiple task replay buffers. During each replay update, a fixed number of tasks (less or equal to the total number of available training tasks) are randomly selected, then an equal number of samples are drawn for each task to construct the replay batch.

4.2 Test-time Adaptation Setup

Both MT-C2F-ARM and Reptile-C2F-ARM use the same C2F-ARM update and adapt the agent parameters to the new task via gradient descent. Adaptation for PEARL-C2F-ARM is done by gathering rollout samples in the new environment and re-computing the context embeddings, hence running only inference on the agent's policy model.

4.3 Results

The first set of evaluations are the most challenging for adaptation: an unseen **test-time task** given **0 demonstrations**. The agent is expected to leverage knowledge and skills gained in the 10 training tasks and perform intelligent exploration on the test task, without any guidance from demonstrations. Results for this setup are presented in the top row of Figure 6. Across all 5 test tasks, multi-task fine-tuning performs equally as well as Reptile while performing significantly better than both PEARL and training from scratch.

We next investigate the effect of reward sparsity on test-time performance. We now provide test-time demonstrations of each of the methods, as an aid for exploration under sparse reward. Results in the second and third row of Figure 6 show how the methods behave when given 1 and 2 *test-time demonstrations*. The fact that increasing the number of demonstrations improves training from scratch performance is unsurprising, however, one intriguing observation is that this effect is less apparent for MT-C2F-ARM and Reptile-C2F-ARM methods. This is encouraging evidence that **fine-tuning significantly reduces (or even omit) the need for demonstrations in sparse rewarded tasks, with little loss to performance.** We further investigate the various properties of fine-tuning C2F-ARM in Appendix D.

Apparent from Figure 6 is that PEARL does not seem equipped to handle such a disjoint train-test split. Recall that PEARL adapts without model parameter updates, and the only way to understand a new task is via aggregating new experiences into the context. However, the context encoder clearly fails at providing a useful context for the unseen tasks. we hypothesize this is due to our tasks setup: the training tasks are so visually disjoint that the agent never needs to learn high-quality context embeddings to infer which task it should do. This is different from the original experiment setup in PEARL, where *variations* are treated as "tasks", meaning that the observations from different "tasks" are similar or even identical; in order to disentangle the correct task to perform, the network is heavily motivated to read the context. Further evidence towards this hypothesis can be seen by looking at the zero-shot performance of the PEARL agent (i.e., environment steps $= 0$), where it starts with the same performance as multi-task and Reptile agents but doesn't improve. This suggests the meaningful performance that PEARL does achieve should be attributed to the pretraining and not test-time adaptation.

5 Atari Experiments

The Arcade Learning Environment (ALE, or the Atari benchmark) is one of the most commonly used environments within the RL community; however, rather than using the benchmark to train individual agents on each game, we use it to study meta-RL. We classify Atari as the hardest benchmark in this study due to the fact that the embodiment, objects, and goals, all vary across tasks; i.e., there is very little overlap between two different games.

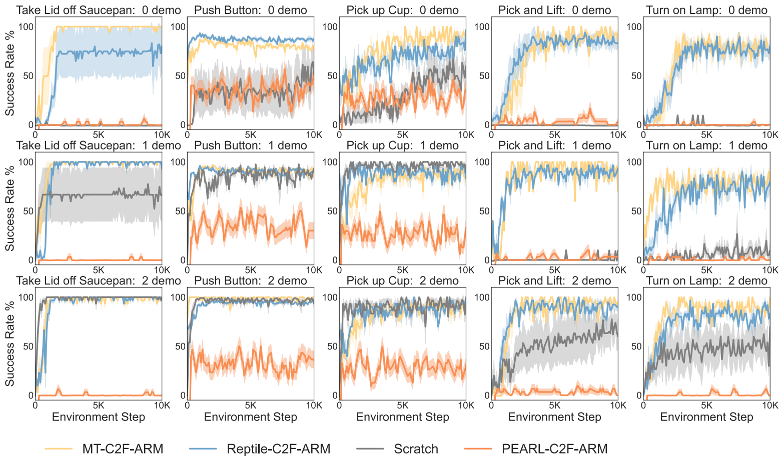


Figure 6: When varying the number of test-time demonstrations (from 0-2), does multi-task pretraining and fine-tuning outperform meta-RL methods on unseen tasks? We perform a “cross-validation” style evaluation: from a set of 11 RLBench tasks, 1 is held out for test-time evaluation, while the other 10 are used for training (and are given 5 demos). This is done for each of the 5 tasks above. Multi-task pretraining and fine-tuning perform equally as well or better than meta-RL. Unsurprisingly, fine-tuning (from either the Multi-task or Reptile agent) requires fewer demonstrations than training from scratch. Solid lines are average over 5 seeds, with shaded regions representing standard errors

Prior work has studied transfer between Atari games on relatively small scales, such as training an RL agent on **one** game and transfer to a **visually similar** game via fine-tuning [32, 33]. To the best of our knowledge, no prior meta-RL algorithm has been meta-trained on a subset of Atari games and tested on other games. Recently, *Oh et al.* [34] demonstrated meta-training via discovering RL update rules can transfer from a mini-grid environment to Atari, but performance still falls short from state-of-the-art results on training Atari games from scratch.

We use the data-efficient version of RainbowDQN [20] as the base RL algorithm, i.e. we use the same training hyper-parameters and image encoder network architecture as proposed in [35]. Note that RL^2 is excluded from this section because it is on-policy.

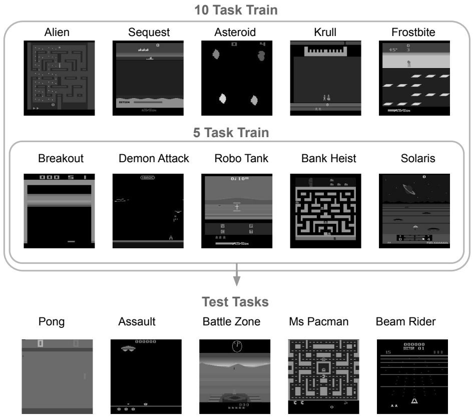


Figure 7: In Atari experiments, each agent is trained on up to 10 tasks and evaluated on 5 unseen tasks.

5.1 Training Setup

Reptile-Rainbow incorporates Reptile [11] with the off-policy batch update in Rainbow. Similar to Reptile-C2F-ARM, transitions from each Atari game are stored in a separate replay buffer. For multiple steps in the inner loop, the agent draws a batch from a randomly sampled buffer and performs updates to the distributional value network. In the outer loop, the value network gets a soft update to mix parameters before and after the inner loop updates.

PEARL-Rainbow conditions a context embedding to the value network. During each batch update, samples from each task are drawn from a recent window of agent experiences and get encoded as the context. The context encoder is additionally trained with a KL loss [12].

MT-Rainbow jointly trains RainbowDQN on all training games/tasks. In contrast to Reptile-Rainbow, the replay batch in both MT-Rainbow and PEARL-Rainbow contains samples from multiple task replay buffers. During each replay update, we randomly select a fixed number of tasks (less or equal to the total number of available training tasks), then draw an equal number of samples for each task and construct the replay batch.

Note that, Atari games generally have different action dimensions, hence for all pretrained agents, the action space is padded to be the maximum possible size (discrete, 18-dimensional). The extra dimensions are all mapped to "No-op" for the games with smaller original action space. For training from scratch, the agents are trained on the original action dimensions on their corresponding test game.

5.2 Test-time Adaptation Setup

We use 5 test-time tasks, namely Pong, Assault, Battle Zone, Ms Pacman, and Beam Rider. For Reptile-Rainbow and MT-Rainbow, a trained agent is finetuned with Rainbow for 100,000 environment steps on each unseen game, following the data-efficient benchmark proposed in [35]. For PEARL-Rainbow, the agent is rolled-out on a new game for 100,000 environment steps, and only updates the context embedding by encoding newly collected experiences.

For both Reptile-Rainbow and MT-Rainbow, all network parameters are finetuned. We provide additional results in Appendix E for loading only the convolutional encoder and re-initializing all MLP layers during finetuning. This setup was previously investigated in [33] at a smaller scale, i.e. training a Breakout agent and fine-tuning on Pong, but the network architecture and base RL algorithms differ from ours.

5.3 Results

We report the test-time adaptation performance for each pretrained agent in Figure 8. The results are divided into 3 sets, based on the number of training tasks provided to each compared algorithm. For each set of experiments, the same training from "scratch" baseline is used (colored in gray).

For 5-task- and 10-task- trained agents, we observe high variance across different test-time games despite being finetuned from the same pretrained agent. For example, finetuning the Reptile-Rainbow agent trained on 10 tasks provides a clear advantage over training from scratch on Assault, but falls short on Ms Pacman and Beam rider.

Consistent with results on RLBench, PEARL-Rainbow also fails to adapt to the unseen task despite continuously updating its context embedding for a sufficiently long period. PEARL-Rainbow's zero-shot performance on some of the games is clearly better than random (i.e. the first data point for training from scratch)— however, this benefit comes more from the diversity of training tasks than from the meta-training algorithm itself, which is evident from observing that: **(1)** PEARL-Rainbow has similar zero-shot performance to that of MT-Rainbow and Reptile-Rainbow, and **(2)** as the number of training task goes from 5 to 10, PEARL-Rainbow's performance slightly improves. Overall, training from scratch has competitive performance across all test-time tasks, which is in contrast to previous results on Procgen and RLBench. Previous work has also found that transfer between Atari games is challenging [36, 32, 37], and the common hypothesis is that the visuals and control strategies vary too much across games for positive transfer. We provide further empirical results in Appendix E to gain more insights into this hypothesis.

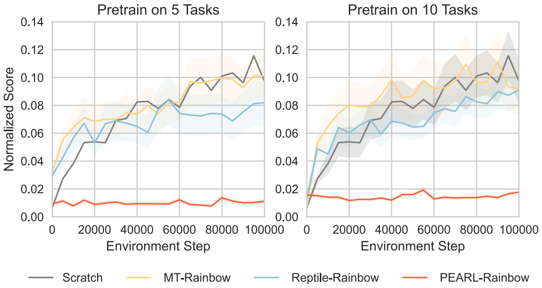


Figure 8: Adaptation on unseen Atari games. Solid lines are average over 10 seeds, with shaded regions representing standard errors.

6 Related Work

Meta-Reinforcement Learning Meta-RL aims to find the best learning strategy that enables fast adaptation to a new task via reinforcement learning. This often relies on meta-training with a distribution of tasks and exploiting their shared structures. Two main approaches include context-based methods and gradient-based methods. Context-based methods are trained to use recent rollout experiences from a new task to form a context that can be used to distinguish what task the policy is solving. Previously, this context has been formed implicitly via an LSTM [16, 38], or explicitly, by passing trajectories through a separate encoder, whose output is given to a context-conditioned policy [12, 39]. Gradient-based methods perform test-time optimisation of hyperparameters [40], loss functions [41, 42], or network parameters [3, 43, 44].

The meta-RL approaches above have only been studied in fully observable state settings with shaped rewards; neglecting more realistic real-world scenarios, where rewards are often sparse, and observations are high-dimensional (e.g. images, point clouds, etc). There is limited work that study these issues: for example, hindsight relabeling is used to aid in sparse reward setups e.g. [45], but uses fully observable states. Other approaches to sparse reward and partial observability include HyperX [46], DREAM [47], and MetaCure [48]. Out-of-distribution variation adaptation within the same task is another challenging setup, where recent methods include model-identification, experience relabeling, [49], and adding symmetries [50].

Beyond context-based and gradient-based methods built on model-free RL algorithms, other lines of work include: **model-based meta-RL**, via meta-training a dynamics model and has seen success in enabling adapting to different hardware or terrain conditions on a legged millirobot [51]; **meta-imitation learning**, has been applied to vision-based robot manipulation [10, 52, 53]; **meta-learn RL algorithms**, which aims to discover RL objectives or update rules that can be transferred across different task environments [34, 54–57].

Although in our experiments, we follow the original designs pf PEARL and RL^2 which don't allow gradient updates during test time, recent work [58] has looked into the theoretical limitations of context-based meta-RL algorithms in out-of-distribution variation adaptation setting [58], and shown that adding gradient updates (i.e. finetuning) at test time helps improve adaptation.

Multi-task Reinforcement Learning The pretraining procedure in our experiments is multi-task reinforcement learning (MTRL), where the training objective is simply finding a single best policy across multiple tasks. The main challenge in multi-task learning in general lies in multi-objective optimization, and has been investigated in MTRL [59, 60] and applied to robotics [61]. Recently, *Kurin et al.* [62] demonstrated that joint training with proper regularization achieves competitive performance with the more complicated multi-task algorithms. This observation aligns with our multi-task training results.

Meta- v.s. Multi-task pretraining in RL Multi-variation pretraining followed by fine-tuning, also called domain random search (DRS), is also shown to achieve comparable performance to meta-RL on existing state-based benchmarks [63]. Our work expands on this setup by training on more distinct **tasks** instead of variations, and excluding the test-time task from training. Notably, the meta-learning suite (e.g. ML10, ML45) in the MetaWorld [64] benchmark also poses such task generalization challenges, and finetuning is recently shown to be better than meta-RL algorithms such as RL^2 and MAML [24].

7 Conclusion

We perform a large-scale study on vision-based meta-RL across a *truly* diverse set of tasks. Our results show that when trained and tested on truly diverse reinforcement learning tasks, simple pretraining and fine-tuning can perform equally as well as well as more complex meta-learning approaches. This is consistent with the findings within the computer vision community [4–7], but in contrast to the large body of current literature, which shows that meta-RL is effective when evaluated on *variations* of the same train-time task.

Our work is an initial step towards understanding the subtleties between meta-RL and multi-task pretraining, with plenty of room for further investigation. Despite the breadth of our experiments across 3 benchmarks, a limitation of our study is the limited number of pre-training tasks for both RLBench and Atari. This may not be sufficient to fully learn the representations required to enable the best adaptation performance on new tasks. However, note that the purpose of this study was not to show that multi-task pretraining followed by fine-tuning is better than training from scratch, but rather that multi-task pretraining can perform equally as well, or better, than meta-RL.

Acknowledgments and Disclosure of Funding

This work was supported by DARPA RACER and Hong Kong Centre for Logistics Robotics. The authors would like to thank Shikun Liu, Danijar Hafnar, Olivia Watkins, Yuqing Du, and Luisa Zintgraf for their help and feedback on initial drafts of the paper.

References

[1] Vinyals, O., C. Blundell, T. Lillicrap, et al. Matching networks for one shot learning. *Advances in Neural Information Processing Systems*, 29:3630–3638, 2016.

[2] Ravi, S., H. Larochelle. Optimization as a model for few-shot learning. *Intl. Conference on Learning Representations*, 2017.

[3] Finn, C., P. Abbeel, S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Intl. Conference on Machine Learning*, pages 1126–1135. PMLR, 2017.

[4] Chen, W.-Y., Y.-C. Liu, Z. Kira, et al. A closer look at few-shot classification. *Intl. Conference on Learning Representations*, 2019.

[5] Dhillon, G. S., P. Chaudhari, A. Ravichandran, et al. A baseline for few-shot image classification. *Intl. Conference on Learning Representations*, 2020.

[6] Tian, Y., Y. Wang, D. Krishnan, et al. Rethinking few-shot image classification: a good embedding is all you need? In *European Conference on Computer Vision*, pages 266–282. Springer, 2020.

[7] Chen, Y., Z. Liu, H. Xu, et al. Meta-baseline: exploring simple meta-learning for few-shot learning. In *Intl. Conference on Computer Vision*, pages 9062–9071. 2021.

[8] James, S., Z. Ma, D. Rovick Arrojo, et al. RLBench: The robot learning benchmark & learning environment. *IEEE Robotics and Automation Letters*, 2020.

[9] Mandi, Z., F. Liu, K. Lee, et al. Towards more generalizable one-shot visual imitation learning. *arXiv preprint arXiv:2110.13423*, 2021.

[10] Finn, C., T. Yu, T. Zhang, et al. One-shot visual imitation learning via meta-learning. In *Conference on Robot Learning*, pages 357–368. PMLR, 2017.

[11] Nichol, A., J. Achiam, J. Schulman. On first-order meta-learning algorithms. *arXiv preprint arXiv:1803.02999*, 2018.

[12] Rakelly, K., A. Zhou, D. Quillen, et al. Efficient off-policy meta-reinforcement learning via probabilistic context variables, 2019.

[13] Cobbe, K., C. Hesse, J. Hilton, et al. Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*, pages 2048–2056. PMLR, 2020.

[14] Bellemare, M. G., Y. Naddaf, J. Veness, et al. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[15] Machado, M. C., M. G. Bellemare, E. Talvitie, et al. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018.

[16] Duan, Y., J. Schulman, X. Chen, et al. rl^2: Fast reinforcement learning via slow reinforcement learning. *arXiv preprint arXiv:1611.02779*, 2016.

[17] Cachet, T., J. Perez. Transformer-based meta-imitation learning for robotic manipulation. 2020.

[18] Schulman, J., F. Wolski, P. Dhariwal, et al. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[19] James, S., K. Wada, T. Laidlow, et al. Coarse-to-Fine Q-attention: Efficient Learning for Visual Robotic Manipulation via Discretisation. *IEEE Conference on Computer Vision and Pattern Recognition*, 2022.

[20] Hessel, M., J. Modayil, H. Van Hasselt, et al. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-second AAAI conference on artificial intelligence*. 2018.

[21] Wang, K., B. Kang, J. Shao, et al. Improving generalization in reinforcement learning with mixture regularization. *Advances in Neural Information Processing Systems*, 33:7968–7978, 2020.

[22] Yarats, D., R. Fergus, A. Lazaric, et al. Mastering visual continuous control: Improved data-augmented reinforcement learning. *arXiv preprint arXiv:2107.09645*, 2021.

[23] Igl, M., G. Farquhar, J. Luketina, et al. Transient non-stationarity and generalisation in deep reinforcement learning, 2020.

[24] Anand, A., J. Walker, Y. Li, et al. Procedural generalization by planning with self-supervised world models. *arXiv preprint arXiv:2111.01587*, 2021.

[25] Alver, S., D. Precup. A brief look at generalization in visual meta-reinforcement learning. *arXiv preprint arXiv:2006.07262*, 2020.

[26] Espeholt, L., H. Soyer, R. Munos, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International Conference on Machine Learning*, pages 1407–1416. PMLR, 2018.

[27] Lee, S., S.-Y. Chung. Improving generalization in meta-rl with imaginary tasks from latent dynamics mixture. *ArXiv*, abs/2105.13524, 2021.

[28] Lillicrap, T. P., J. J. Hunt, A. Pritzel, et al. Continuous control with deep reinforcement learning. *Intl. Conference on Learning Representations*, 2015.

[29] Fujimoto, S., H. Van Hoof, D. Meger. Addressing function approximation error in actor-critic methods. *Intl. Conference on Machine Learning*, 2018.

[30] Haarnoja, T., A. Zhou, K. Hartikainen, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[31] James, S., A. J. Davison. Q-attention: Enabling Efficient Learning for Vision-based Robotic Manipulation. *IEEE Robotics and Automation Letters*, 2022.

[32] Sobol, D., L. Wolf, Y. Taigman. Visual analogies between atari games for studying transfer learning in rl. *arXiv preprint arXiv:1807.11074*, 2018.

[33] Du, Y., V. Gabriel, J. Irwin, et al. Initial progress in transfer for deep reinforcement learning algorithms. In *Proceedings of deep reinforcement learning: frontiers and challenges workshop, New York City, NY, USA*. 2016.

[34] Oh, J., M. Hessel, W. M. Czarnecki, et al. Discovering reinforcement learning algorithms. *Advances in Neural Information Processing Systems*, 33:1060–1070, 2020.

[35] van Hasselt, H. P., M. Hessel, J. Aslanides. When to use parametric models in reinforcement learning? *Advances in Neural Information Processing Systems*, 32, 2019.

[36] Rusu, A. A., N. C. Rabinowitz, G. Desjardins, et al. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

[37] Mittel, A., P. 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.

[38] Wang, J. X., Z. Kurth-Nelson, D. Tirumala, et al. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016.

[39] Fakoor, R., P. Chaudhari, S. Soatto, et al. Meta-q-learning, 2020.

[40] Xu, Z., H. van Hasselt, D. Silver. Meta-gradient reinforcement learning. *arXiv preprint arXiv:1805.09801*, 2018.

[41] Sung, F., L. Zhang, T. Xiang, et al. Learning to learn: Meta-critic networks for sample efficient learning. *arXiv preprint arXiv:1706.09529*, 2017.

[42] Houthooft, R., R. Y. Chen, P. Isola, et al. Evolved policy gradients. *arXiv preprint arXiv:1802.04821*, 2018.

[43] Rothfuss, J., D. Lee, I. Clavera, et al. Promp: Proximal meta-policy search. *Intl. Conference on Learning Representations*, 2019.

[44] Raghu, A., M. Raghu, S. Bengio, et al. Rapid learning or feature reuse? towards understanding the effectiveness of maml, 2020.

[45] Packer, C., P. Abbeel, J. E. Gonzalez. Hindsight task relabelling: Experience replay for sparse reward meta-rl. *Advances in Neural Information Processing Systems*, 34, 2021.

[46] Zintgraf, L. M., L. Feng, C. Lu, et al. Exploration in approximate hyper-state space for meta reinforcement learning. In *International Conference on Machine Learning*, pages 12991–13001. PMLR, 2021.

[47] Liu, E. Z., A. Raghunathan, P. Liang, et al. Decoupling exploration and exploitation for meta-reinforcement learning without sacrifices. *arXiv preprint arXiv:2008.02790*, 2020.

[48] Zhang, J., J. Wang, H. Hu, et al. Metacure: Meta reinforcement learning with empowerment-driven exploration. In *International Conference on Machine Learning*, pages 12600–12610. PMLR, 2021.

[49] Mendonca, R., X. Geng, C. Finn, et al. Meta-reinforcement learning robust to distributional shift via model identification and experience relabeling. *arXiv preprint arXiv:2006.07178*, 2020.

[50] Kirsch, L., S. Flennerhag, H. van Hasselt, et al. Introducing symmetries to black box meta reinforcement learning. *arXiv preprint arXiv:2109.10781*, 2021.

[51] Nagabandi, A., I. Clavera, S. Liu, et al. Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. *arXiv preprint arXiv:1803.11347*, 2018.

[52] James, S., M. Bloesch, A. J. Davison. Task-embedded control networks for few-shot imitation learning. In *Conference on Robot Learning*, pages 783–795. PMLR, 2018.

[53] Bonardi, A., S. James, A. J. Davison. Learning one-shot imitation from humans without humans. *IEEE Robotics and Automation Letters*, 5(2):3533–3539, 2020.

[54] Kirsch, L., S. van Steenkiste, J. Schmidhuber. Improving generalization in meta reinforcement learning using learned objectives. *arXiv preprint arXiv:1910.04098*, 2019.

[55] Xu, Z., H. P. van Hasselt, M. Hessel, et al. Meta-gradient reinforcement learning with an objective discovered online. *Advances in Neural Information Processing Systems*, 33:15254–15264, 2020.

[56] Co-Reyes, J. D., Y. Miao, D. Peng, et al. Evolving reinforcement learning algorithms. *arXiv preprint arXiv:2101.03958*, 2021.

[57] Houthooft, R., Y. Chen, P. Isola, et al. Evolved policy gradients. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, R. Garnett, eds., *Advances in Neural Information Processing Systems*, vol. 31. Curran Associates, Inc., 2018.

[58] Xiong, Z., L. M. Zintgraf, J. Beck, et al. On the practical consistency of meta-reinforcement learning algorithms. *ArXiv*, abs/2112.00478, 2021.

[59] Yang, R., H. Xu, Y. Wu, et al. Multi-task reinforcement learning with soft modularization. *ArXiv*, abs/2003.13661, 2020.

[60] Sodhani, S., A. Zhang, J. Pineau. Multi-task reinforcement learning with context-based representations, 2021.

[61] Kalashnikov, D., J. Varley, Y. Chebotar, et al. Mt-opt: Continuous multi-task robotic reinforcement learning at scale, 2021.

[62] Kurin, V., A. De Palma, I. Kostrikov, et al. In defense of the unitary scalarization for deep multi-task learning. *arXiv preprint arXiv:2201.04122*, 2022.

[63] Gao, K., O. Sener. Modeling and optimization trade-off in meta-learning. *Advances in Neural Information Processing Systems*, 33:11154–11165, 2020.

[64] Yu, T., D. Quillen, Z. He, et al. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning, 2021.

[65] Raffin, A., A. Hill, A. Gleave, et al. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

[66] Arulkumaran, K. Pytorch implementation of rainbowdqn. https://github.com/Kaixhin/Rainbow, 2022.

[67] James, S., P. Abbeel. Coarse-to-Fine Q-attention with Learned Path Ranking. *arXiv preprint arXiv:2204.01571*, 2022.

[68] James, S., P. Abbeel. Coarse-to-Fine Q-attention with Tree Expansion. *arXiv preprint arXiv:2204.12471*, 2022.

A Detailed justifications for the checklist questions

1(a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?[Yes] We believe our experiments provide an extensive study to our problem statement, comparing multi-task pretraining and finetuning with meta-RL. The empirical results are comprehensive and clearly supports our stated contributions.

(b) Did you describe the limitations of your work? [Yes] Yes, see section 7

(c) Did you discuss any potential negative societal impacts of your work?[Yes] Yes, see section 7

(d) Have you read the ethics review guidelines and ensured that your paper conforms to them?[Yes]

2. If you are including theoretical results...[N/A]

3. If you ran experiments...

(a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Please refer to the code scripts provided in the supplemental material.

(b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] Please refer to both main text and appendix for experiment details.

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)?[Yes]

All adaptation experiments in Procgen and RLBench are run for 3 seeds. In Atarim all adaptation experiments are run for 10 seeds due to the high variance during RL training.

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]

As stated in section 2, we use RTX A5000 GPUs each with 24GB memory. Procgen experiments are all done on single-GPU: it took 10 hours for training each multi-task agent for 100M environment steps, and 10 minutes for each finetuning/adaptation run on each of the 20 test levels. RLBench experiments use 4 GPUs for each run; training each multi-task agent took 24 hours, and finetuning/adaptation on each test task for each of the 3 seeds takes 9 hours. Atari experiments are done on single GPUs; training each agent takes 24hrs for 1 million steps, and finetuning/adaptation on each test game took 1 hours for each of the 10 seeds.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes]

We use the Procgen RL environment released by OpenAI under MIT lisence [13]; the RL training code is built on the open-source RL framework implementation from Stable-baselines3 [65]

We use the RLBench simulated robotic manipulation environment [8] released under MIT license. Our C2F-ARM algorithm and training framework are built based on the original author's implementation and open-sourced code under MIT license.

We use The Arcade Learning Environment [15, 14] (under MIT license) for simulated RL environments in our Atari experiments. The code for RainbowDQN is built on the open-source implementation [66]

(b) Did you mention the license of the assets? [Yes] See described above, all used assets are released under MIT license.

(c) Did you include any new assets either in the supplemental material or as a URL? [Yes] Yes, please see the supplemental material for all the code for reproducing our experiments.

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects... [N/A]

B Summary of Compared Algorithms

In this section, we illustrate a unified overview of how the meta-RL algorithms are implemented in Algorithms 1 and 2.

Algorithm 1 Training On-policy Meta-RL

```
Require: Learning rate $\alpha$; Reptile step size $\epsilon$;
Policy optimization objective $\mathcal{L}$; Trajectory buffer $\mathcal{D}$
Input: Algorithm name $algo$
Initialize policy $\pi_\theta$, LSTM embedder $\phi$
while not converged do
    if $algo == $ Reptile then
        Sample a single task $\mathcal{T} \sim p(\mathcal{T})$
        Set $\theta^{old} \leftarrow \theta$
        for $i = 0$ to $k - 1$ do
            Collect trajectories $\mathcal{D}^i$ using $\pi_{\theta^i}$
            $\theta^{i+1} \leftarrow \theta^i - \alpha \nabla_{\theta^i} \mathcal{L}(\pi_{\theta^i}, \mathcal{D}^i)$
        end for
        $\theta \leftarrow \epsilon(\theta^{old} - \theta^k)$
    else if $algo == $ RL$^2$ then
        for $i = 1$ to $b$ do
            Sample task $\mathcal{T}_i \sim p(\mathcal{T})$; collect $\mathcal{D}^i$ using $\pi_{\theta^i}, \phi$
        end for
        $\theta \leftarrow \theta^i - \alpha \nabla_\theta \sum_{i=1}^{b} \mathcal{L}^i(\pi_\theta, \phi, \mathcal{D}^i)$
        $\phi \leftarrow \theta^i - \alpha \nabla_\phi \sum_{i=1}^{b} \mathcal{L}^i(\pi_\theta, \phi, \mathcal{D}^i)$
    else if $algo == $ Pretrain then
        for $i = 1$ to $b$ do
            Sample task $\mathcal{T}_i \sim p(\mathcal{T})$; collect $\mathcal{D}^i$ using $\pi_\theta$
        end for
        $\theta \leftarrow \theta - \alpha \nabla_\theta \sum_{i=1}^{b} \mathcal{L}(\pi_\theta, \mathcal{D})$
    end if
end while
```

Algorithm 2 Training Off-policy Meta-RL

```
Require: Learning rate $\alpha$; Reptile step size $\epsilon$
Context sampling heuristics $\mathcal{S}_c$; Value-function loss $J_Q$;
KL loss $\mathcal{L}_{KL}$;
Input: algorithm name $algo$
Initialize Q-function $Q_\theta$, context encoder $\phi$;
while not converged do
    Sample $B$ replay buffers and draw samples from each buffer to
    construct batch $b = \{b^1, \ldots, b^j\}$
    if $s == $ Reptile then
        for $j = 1$ to $B$ do
            $\theta_j^{old} \leftarrow \theta$
            for $i = 0$ to $k - 1$ do
                $\theta^{i+1} \leftarrow \theta^i - \alpha \nabla_{\theta^i} J_Q(b^j)$
            end for
            $\theta \leftarrow \epsilon(\theta^{old} - \theta_j^k)$
        end for
    else if $algo == $ PEARL then
        for $j = 1$ to $B$ do
            Sample contexts $c^j = \mathcal{S}_c(\mathcal{D}_i)$
            $\mathcal{L}_{KL}^j = \mathcal{L}_{KL}(\phi(c^j))$
        end for
        $\phi \leftarrow \alpha \nabla_\phi \sum \mathcal{L}_{KL}^j$
        $\theta \leftarrow \theta - \alpha \nabla_\theta J_Q(b, \{c^1, \ldots, c^j\})$
    else if $s == $ Pretrain then
        $\theta \leftarrow \theta - \alpha \nabla_\theta J_Q(b)$
    end if
end while
```

C Additional Procgen Remarks

C.1 Experiment Details

Hyperparameters See Table 1 for hyperparameter settings used for RL training on Procgen. The base PPO training parameters are shared across all compared algorithms. Our code is built off the PPO training code implemented in Stable-baselines 3 [65].

Task settings See Figure 10 for additional visualizations of the training levels. We show 10 levels from each training set, note that each smaller training set is a subset of levels from the bigger training sets. All 20 test levels are shown under "test levels". Note the high diversity of color themes and layouts across different levels.

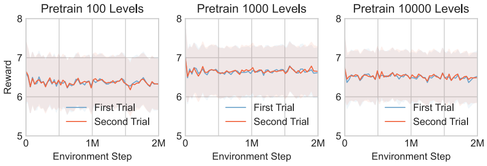


Figure 9: Episodic rewards from first and second (last) trial achieved by RL^2 agents when adapting to unseen test levels. No clear different between the two trial suggests that the adaptation behavior fails to use previous episodes' exploratory knowledge and increase performance at later episode

C.2 Training Results for Procgen Experiments

The results on training levels for all compared methods are reported in Table C.2, where each column corresponds to training each agent with a different size of training set, i.e. the number of training-levels used. Each cell reports the final reward for a single agent that was trained for 100M environment steps, averaged over all training levels.

Method	Average Reward on 100 levels	Average Reward on 1000 levels	Average Reward on 10000 levels
MT-PPO	9.7	8.0	7.5
RL^2-PPO	4.7	7.7	7.8
Reptile-PPO	9.5	5.8	5.3

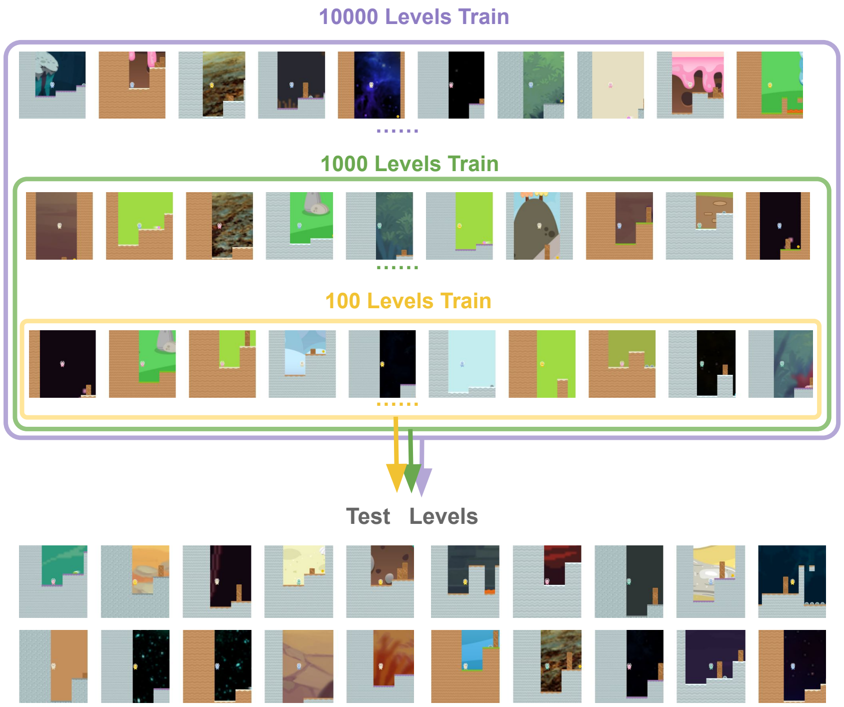


Figure 10: Visualization of all 20 test levels and a subset of training levels from game Coinrun in Procgen.

C.3 Additional Results

How well does RL^2 adapt See Figure 9 for a comparison between the performance of the first rollout and the second rollout at test time. Results show that across all types of training task setups, the first trial has similar rewards than the second trial, suggesting a failure to adapt.

D Additional RLBench Remarks

D.1 Coarse-to-fine Attention Driven Robotic Manipulation (C2F-ARM)

A core component of C2F-ARM is the coarse-to-fine Q-attention [19] module, which takes as input a coarse 3D voxelization of the scene, and learns to attend to interesting areas within the coarse voxelization. The module then 'zooms' into that area and re-voxelizes the scene at a higher spatial resolution. This 'attend-and-zoom' procedure is applied iteratively and results in a near-lossless discretization of the 3D space. This 3D space discretization is combined with a rotation discretization to give a continuous next-best pose output. C2F-ARM takes this next-best pose and uses a motion planner to take the robot to the goal pose. In this work, we use the original C2F-ARM algorithm, and do not include any of its subsequent extensions, e.g., learned path ranking [67] and tree expansion [68].

As mentioned above, a small number of demonstrations are used to overcome the exploration problem within these sparse-reward environments. The predecessor to C2F-ARM, ARM [31], introduced two demonstration pre-processing procedures: (1) *keyframe discovery*, which assists the Q-attention at the initial phase of training by suggesting meaningful points of interest; and (2) *demo augmentation*, which takes demo episodes and produces many sub-episodes with different starting points, thereby increasing the initial number of demos in the replay buffer. All C2F-ARM results use both pre-processing procedures.

D.2 Training Results for RLBench Experiments

We report results on training tasks for all compared methods in Table D.2, where each row corresponds to training with one task held-out and using only the remaining 10 tasks. Each cell reports the final success rate for a single agent on all training tasks, each evaluated 10 episodes and results are averaged across tasks.

Table 1: Hyperparameters for Procgen Experiments

PPO	Value
Environment steps (pretraining)	100,000,000
Environment steps (testing)	2,000,000
Mini batch size	2048
Learning rate	5e-4
Number of epochs	3
Discount factor γ	0.99
GAE coefficient λ	0.95
Clip Range	Constant 0.2
Entropy coefficient	0.01
Value function coefficient	0.5
Gradient clipping norm	0.5
Target KL divergence	0.01
Reptile-PPO	Value
Number of inner-loop iterations	3
Soft parameter update schedule ϵ	Linear 0-1
RL^2-PPO	Value
LSTM hidden state dimension	256
LSTM number of hidden layers	2

Held-out Task	MT-C2FARM Avg. Success Rate	PEARL-C2FARM Avg. Success Rate	Reptile-C2FARM Avg. Success Rate
Lid off Saucepan	64.64 ± 9.89	49.88 ± 11.45	71.91 ± 10.58
Push Button	60.93 ± 12.29	62.86 ± 9.73	55.97 ± 13.58
Pick and Lift	65.30 ± 10.15	51.19 ± 12.99	68.84 ± 12.67
Pick up Cup	67.30 ± 9.40	53.17 ± 11.00	47.74 ± 11.70
Turn on Lamp	68.63 ± 10.78	57.80 ± 11.21	54.97 ± 12.85

D.3 Single-task, Multi-variation Experiments

As a sanity check for whether the meta-RL algorithms are able to generalize to an easier, multi-variation setup, we experiment with the multi-variation *push_button* task from RLBench, where the task is to push a button on the tabletop but different variations differ in button colors. Each method is trained on 10 variations and average test-time performance on 5 unseen variations. Results are reported in 11: all compared methods can perform an unseen variation in a zero-shot manner, and PEARL is able to adapt despite the lack of gradient updates.

D.4 Properties of fine-tuning Q-attention

Earlier in the paper, we have shown that multi-task pre-training, followed by fine-tuning, can perform equally as well as meta-RL. In the following set of experiments, we explore fine-tuning in more detail, focusing on: **(1)** zero-shot performance on test tasks (no test-time gradient updates); **(2)** investigate whether it is better to fine-tune an unseen task in isolation, or together with other tasks (in a multi-task setup); and **(3**) inspecting the role of each Q-attention depth on fine-tuning performance.

We begin by evaluating zero-shot task performance on held-out test tasks when pretrained with multi-task pretraining and evaluated on 30 episode rollouts. Results in Table 2 show that multi-task pretraining, even on a small number of tasks, can allow the Q-attention to begin to learn an intuition of 'objectness', which can

Table 2: Zero-shot task performance on held-out test tasks, when pretrained with multi-task pretraining. In column "Success Rate (Train)", we report the final training performance averaged across evaluating 30 episodes for each of the 10 training tasks. In column "Success Rate (Unseen Task)", we report zero-shot direct evaluation performance of the trained agent on the held-out unseen task.

Held-out Task	Success Rate (Train)	Success Rate (Unseen Task)
Lid off Saucepan	64.64 ± 9.89	0 ± 0
Push Button	60.93 ± 12.29	63.33 ± 8.80
Pick and Lift	65.30 ± 10.15	0 ± 0
Pick up Cup	67.30 ± 9.40	36.43 ± 8.83
Turn on Lamp	68.63 ± 10.78	3.33 ± 3.28

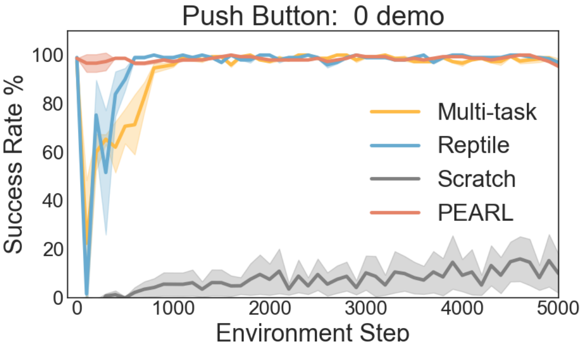


Figure 11: Adaptation result on unseen variation of the “push button" task on RLBench. Each method is trained on 10 variations and average test-time performance on 5 unseen variations. All compared methods can perform an unseen variation in a zero-shot manner, and PEARL is able to adapt despite the lack of gradient updates.

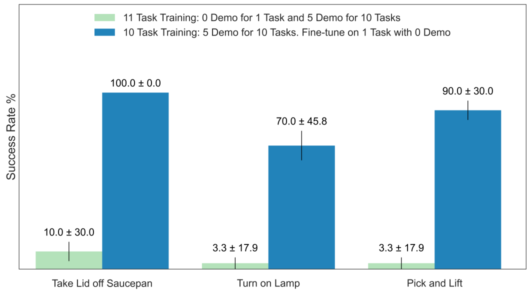


Figure 12: Is it is better to fine-tune an unseen task in isolation, or together with other tasks? We perform a “cross-validation" style evaluation, where from a set of 11 RLBench tasks, 1 is held out for test-time evaluation (and given 0 demos), while the other 10 are used for pretraining (and are given 5 demos). This is done for each of the 3 tasks above. Each color bar represents the average evaluation over 30 episodes while the error bars represent the standard deviations.

be useful for zero-shot performance on some
tasks.

The next set of experiments aims to investigate whether it is better to fine-tune an unseen task in isolation, or together with other tasks (in a multi-task setup). The intuition for the former is that train-time tasks (where we have access to demos), can be used to learn good representations and exploration strategies; while the latter intuition is that mixing with train-task data can act as auxiliary tasks, and the test-time task is treated as the main task. As shown in Figure 12, fine-tuning in isolation is superior to training in a multi-task setting. The hypothesis here is that the agent can keep the representations and skills that are useful to the fine-tune task, while forgetting non-useful ones; whereas training with other tasks requires that the network have the capacity to remember all skills.

The final set of experiments aim to inspect the role of each Q-attention depth on fine-tuning performance. Figure 13 shows a comparison of three different fine-tuning strategies: **(1)** updating only the first (coarse) Q-attention depth, **(2)** updating only the second (fine) depth, and **(3)** updating both depths. Unsurprisingly, updating both depths gives the best performance, however, fine-tuning only the second depth (while leaving the first depth fixed) performs almost equally as well. This suggests that during pretraining, the Q-attention learns a good understanding of ‘objectness’ at the coarse level (i.e., *what* object to interact with), while the “fine" level is more concerned with *how* to interact with the object, which is more task-specific, and therefore has the most benefit from fine-tuning.

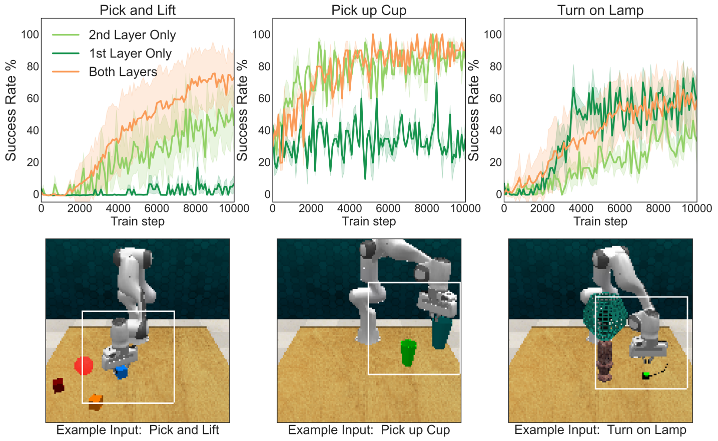


Figure 13: A deeper look at the role of Q-attention when fine-tuning on 3 unseen tasks. **Top:** we compare how performance for 3 different fine-tuning strategies: (1) updating only the first Q-attention depth, (2) updating only the second depth, and (3) updating both depths. **Bottom:** visualization of the scene information captured by the Q-attention; whole image represents Q-attention depth 0 which captures global, whole-scene information, while white square represents Q-attention depth 1 which captures local, fine-grained information.

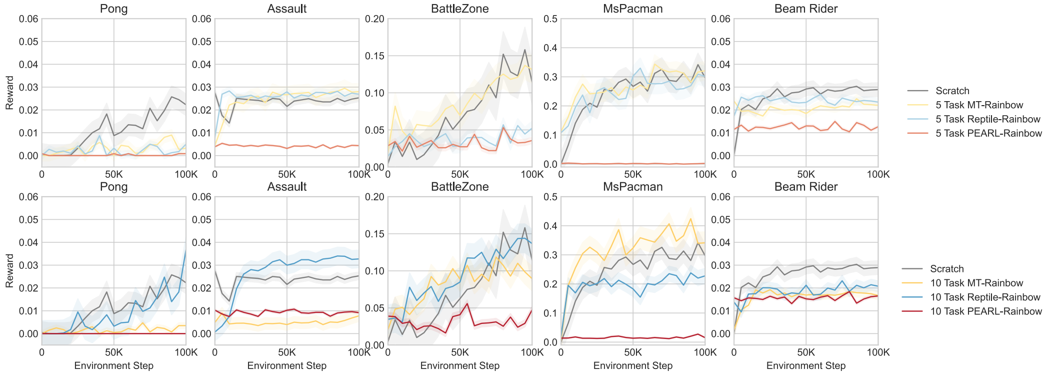


Figure 14: Adaptation results on all 5 test-time Atari games using two different training setups: meta-training or pretraining using 5 tasks (top row) and 10 tasks (bottom row). Each game's scores are normalized by the final converged result for each corresponding game as reported in RainbowDQN [20]. We plot the normalized scores averaged over 10 seeds for each method.

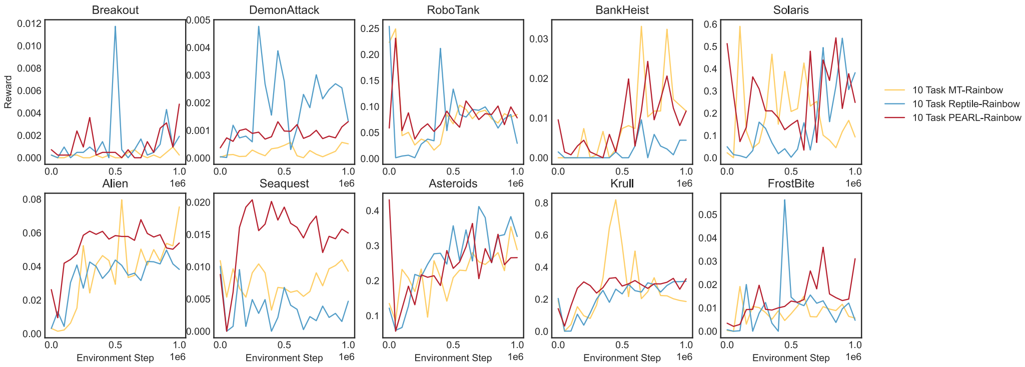


Figure 15: Training performance of the compared method on Atari. Results are reported separately on each training game.

E Additional Atari Remarks

E.1 Full adaptation results

See Figure 14 for separate adaptation results on all 5 test games. For each training method (either training from scratch, adaptation or finetuning), we run 10 seeds on every test game. Because we use the data-efficient benchmark [35] which does not train RainbowDQN to full coverage, we normalize the scores on each game by the corresponding final RainbowDQN result, as reported in [20]. We then plot the normalized reward averaged over all 10 seeds. The first and second row shows adaptation performance from two different training sets: meta-training or pretraining using 5 tasks (top row) versus 10 tasks (bottom row).

Hyperparameters for training base RainbowDQN (data-efficient version) are shared across all compared algorithms, see Table 3 for details. Our code is built on the popular open-source implementation from [66].

E.2 Training Results for Atari Experiments

We report the performance of the three compared method during training in 15 : each method is trained on the same set of 10 Atari games, and rewards are normalized in the same way as the adaptation results, i.e. normalized by the per-game results reported in RainbowDQN [20] (since our setup follows the data-efficient setup in [35], the results are lower than reported in [20]). Notice that, the overall performance of all three methods (MT-Rainbow, Reptile-Rainbow, and PEARL-Rainbow) are similar across each training game, which suggests that their corresponding adaptation performances are not particularly held-back or aided by the performance on training tasks.

E.3 Additional Finetuning Experiments

We compare two ways of finetuning a multi-task agent: finetune all network parameters, versus finetune only the convolutional layers and re-initialize the MLP layers. Results are reported in Figure 17 We note again the small difference between the two sets of results and between finetuning and training from scratch, which further suggests the possibility that there is little shared knowledge between Atari games that can be transferred.

E.4 Finetuning PEARL-Rainbow Experiments

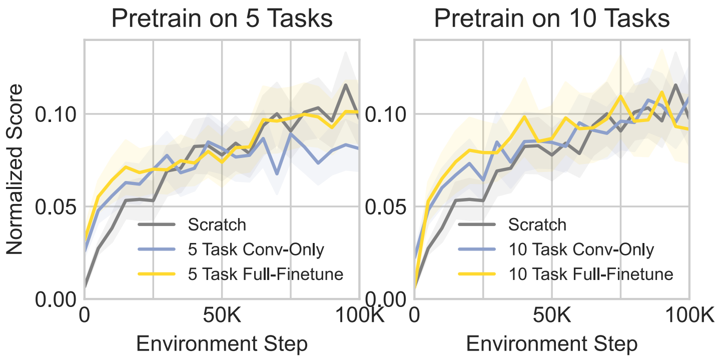


Figure 16: Finetuning the entire network versus finetuning with a partially-loaded (only convolutional layers) agent. Averaged performance across all 5 test-time Atari games shows little difference between the two settings, and shows similar trend when pretrained with different number of tasks (5 v.s. 10). When additionally compared with training from scratch, the results suggest that the transferability between Atari games is likely to be too low for pretraining

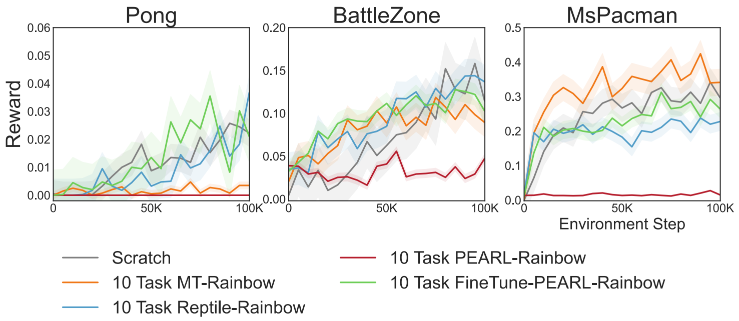


Figure 17: Can PEARL adapt to the new unseen games via finetuning? We finetune a PEARL agent, which was pretrained on 10 Atari games, on 3 separate test-time games. In contrary to gradient-free adaptation (shown in red), finetuning the PEARL-Rainbow agent (shown in green) allows it to adapt to the unseen games, and achieves similar performance to finetuning the multi-task agent and the Reptile agent.

Table 3: Hyperparameters for Atari Experiments

RainbowDQN	Value
Environment steps (pretraining)	1,000,000
Environment steps (testing)	100,000
Batch size	32
Learning rate	1e-3
Minimum of value distribution support	-10
Maximum of value distribution support	10
Multi-step return	20
MLP hidden dimension	256
Action dimension	18
Reptile-Rainbow	Value
Number of inner-loop updates	5
Soft parameter update schedule ϵ	Linear 0-1
PEARL-Rainbow	Value
Context embedding dimension	32
KL loss coefficient	1