Real-Time Legged Locomotion Control with Diffusion from Offline Datasets

Ruofeng Wang

May 17, 2024

Acknowledgement

I would like to express my sincere gratitude to Professor Borivoje Nikolic and Professor Sophia Shao for their invaluable support and guidance throughout the duration of this project. Their expertise and insights have been fundamental to my development and the success of this work.
I am also grateful to Xiaoyu Huang and Yufeng Chi for their collaboration. Working together on the extensive experiments and exploring the diverse fields of robotics, machine learning, and hardware-software acceleration has been a profoundly enriching experience.
This endeavor required us to venture into unfamiliar territories and deeply engage with a broad range of topics. I am thankful for the team's commitment and perseverance as we pursued our shared objectives.

Real-Time Legged Locomotion Control with Diffusion from Offline Datasets

by

Ruofeng Wang

A thesis submitted in partial satisfaction of the

requirements for the degree of

Master of Science

in

Electrical Engineering and Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Borivoje Nikolic, Chair
Professor Sophia Shao

Spring 2024

The thesis of Ruofeng Wang, titled Real-Time Legged Locomotion Control with Diffusion from Offline Datasets, is approved:

Chair [illegible] Date May 16, 2024.

Sophia Shao Date 05/17/2024

University of California, Berkeley

Real-Time Legged Locomotion Control with Diffusion from Offline Datasets

Abstract

Real-Time Legged Locomotion Control with Diffusion from Offline Datasets

by

Ruofeng Wang

Master of Science in Electrical Engineering and Computer Science

University of California, Berkeley

Professor Borivoje Nikolic, Chair

This work introduces DiffuseLoco, a framework for training multi-skill diffusion policies for dynamic legged locomotion from offline datasets, which then enables real-time control of robots in the real world. Learning multiple locomotion skills within a single policy presents a significant challenge in legged control. To address this, offline learning from multi-modal datasets with diffusion models can yield a policy with a rich distribution of locomotion skills. However, due to the larger-scale model and iterative denoising process, diffusion models have their own limitations in achieving real-time control onboard the robot. DiffuseLoco is developed to tackle these issues, utilizing several improvements such as goal-conditioning, receding horizon control, delayed inputs, and an accelerated computing pipeline. We highlight DiffuseLoco with its multi-modality in locomotion skills, zero-shot transfer to real quadrupedal robots, and capability in real-time control on edge computing devices. Through over 200 benchmarking in real-world experiments, DiffuseLoco demonstrates better stability and velocity tracking performance compared to state-of-the-art baselines. The five skills we benchmarked include walking at three different speeds, as well as turning left and right. In the experiments, our DiffuseLoco policy is capable of switching skills smoothly and exhibits robustness against various environments. In addition, we conduct extensive ablation studies to support the design choices in DiffuseLoco. This work opens a new possibility of leveraging imitation learning to create multi-skill controllers for legged locomotion from offline datasets.

Contents

List of Figures

List of Tables

Acknowledgments

I would like to express my sincere gratitude to Professor Borivoje Nikolic and Professor Sophia Shao for their invaluable support and guidance throughout the duration of this project. Their expertise and insights have been fundamental to my development and the success of this work.

I am also grateful to Xiaoyu Huang and Yufeng Chi for their collaboration. Working together on the extensive experiments and exploring the diverse fields of robotics, machine learning, and hardware-software acceleration has been a profoundly enriching experience.

This endeavor required us to venture into unfamiliar territories and deeply engage with a broad range of topics. I am thankful for the team's commitment and perseverance as we pursued our shared objectives.

Chapter 1

Introduction and Background

1.1 Introduction

Data-driven methods for controller development have become increasingly prevalent throughout the legged robotics community. Reinforcement learning (RL) has allowed real legged robots to acquire dynamic and robust locomotion controllers [26, 36, 48]. However, the majority of these RL systems train policies that hone in on a single strategy for solving a given task. While the strategies acquired by these policies may be effective with respect to task performance, they are often not able to capture the rich space of strategies that can be used to solve a given task. For example, in the context of quadrupedal locomotion, a variety of different *skills*, such as trotting and pacing, can be used to move at a target speed [37]. Policies trained using standard RL methods tend to fail to capture these multi-modal solutions.

Transitioning from online RL to offline learning with prior datasets could potentially mitigate this challenge of learning to model multi-modal strategies. Offline learning enables the agent to incorporate the data collected from various skill-specific sources, which can individually model a unimodal solution, but collectively provides a rich multi-modal distribution of possible solutions for a given task. Supervised learning with large-scale offline data has been highly effective in a large variety of domains, such as computer vision and natural language processing, where scaling up both the size of models and datasets leads to better performance and generalization [30, 57]. This has led to the development of powerful generative models, like diffusion models, which are able to model complex multi-modal data distributions [54, 52].

In robotics, imitation learning (IL) [13] from offline datasets has also been shown to be an effective and scalable approach for developing more versatile policies for domains such as robotic manipulation [5, 4] and autonomous driving [9, 16, 46]. However, these domains typically only involve agents that have low-dimensional action spaces (*e.g.*, end-effector trajectory), with low re-planning frequency on inherently stable systems (*e.g.*, robot arms or cars). Additionally, due to the larger scale and especially the iterative denoising process of diffusion models, it is difficult to leverage methods commonly used in these domains

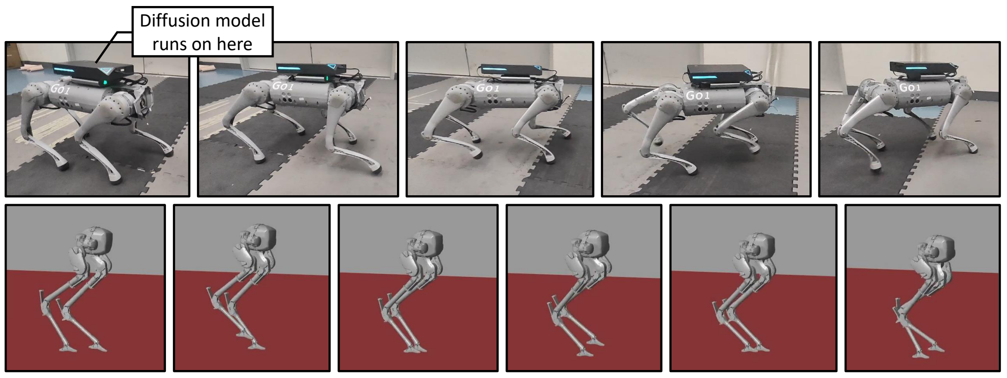

Figure 1.1: Snapshots of (top) a quadrupedal robot Go1 trotting with our diffusion-based real-time locomotion control policy, a DiffuseLoco policy, in the real world with onboard computing, and (bottom) a bipedal robot Cassie controlled by our policy walking in a high-fidelity simulation environment. We present DiffuseLoco, a framework that leverages diffusion models to learn multi-modal legged locomotion control from offline datasets with a specific focus on real-time edge computing for real-world robots. DiffuLoco is able to perform multiple locomotion skills with a single policy, exhibiting robustness in the real world, and is versatile for extending to bipedal locomotion control.

in real-time control systems out-of-box. Running real-time control on edge computing is at another level of difficulty.

In this work, we leverage the expressiveness of diffusion models to learn agile locomotion skills for legged robots from diverse offline datasets and seek to address the aforementioned challenges. We propose *DiffuseLoco*, a framework that leverages diffusion models to learn dynamic legged locomotion control from multi-modal offline datasets. Once trained, our controllers can be effectively deployed on real-world legged robots for real-time control.

1.2 Background and Related Work

Our proposed framework leverages diffusion models for legged locomotion control. In this section, we review the most closely related works on traditional locomotion, learning-based legged locomotion and applications of diffusion models in robotics.

1.2.1 Traditional Locomotion Control

Two of the most popular traditional locomotion control methods are model predictive control (MPC) and proportional-integral-derivative (PID) Controller.

1.2.1.1 Proportional-Integral-Derivative Controller

PID controllers are the backbone of classical control strategies in robotics. They compute control signals by considering the proportional, integral, and derivative terms of the error between the desired and actual state. This method offers simplicity, robustness, and effectiveness in a wide range of applications, from industry control systems to complex humanoid robots.[45] PID controllers excel in systems where high precision and stability are required, ensuring consistent performance even in the presence of disturbances and uncertainties.

1.2.1.2 Model-Predictive Control

MPC represents a more sophisticated approach, where the control action is obtained by solving an optimization problem that predicts the future behavior of the robot over a finite horizon.[2] This approach considers the dynamic model of the robot and can handle multi-variable control problems with constraints on inputs and states. MPC is particularly beneficial in scenarios requiring adaptive and anticipatory control actions, such as in dynamic and unpredictable environments. It offers the advantage of explicitly considering future events and constraints, leading to more efficient and optimized control strategies.

1.2.2 Learning-based Locomotion Control

Reinforcement learning methods for locomotion control can be categorized into online learning and offline learning.

1.2.2.1 Online Learning

Recent advances in model-free online RL have demonstrated promising results in developing locomotion controllers for quadrupedal robots in the real world [48, 24, 19, 39]. However, learning a multi-skill policy with online RL is still a challenge, due to the limited expressivity of commonly used simple action distributions [58].

As a result, when employing online RL for learning multiple skills with a single policy, there is a tendency for policies to overfit to a single skill, failing to adequately learn and incorporate other skills [71, 73].

To counter this, many previous works train separate policies for each skill, and then coordinate these policies through high-level planning [24, 14, 71]. However, this approach introduces its own set of complications. Specifically, it needs reward tuning for each new skill incorporated, which is not scalable. Moreover, transitioning between these individually trained skills presents a challenge in itself [7].

An alternate way to develop a single policy for multi-skill legged control is to leverage adversarial motion priors (AMP) [49], which have shown multiple successful applications to real-world quadruped robots [18, 61, 66].

These methods, however, require conditioning on different commands [18] or skill representations—either through one-hot encoding [61] or latent variables [66]—to switch between skills and mitigate the risk of mode collapse when learning multiple skills concurrently. Without skill representation, as we will show in Sec. 4.2.2, AMP-based methods produce undesirable sim-to-real transfer.

In this work, we aim to build a policy with a more flexible action distribution in the form of a diffusion model, which then allows us to better model multi-modal behaviors, rather than a conditioned distribution leveraged in previous works.

1.2.2.2 Offline Learning

There have been limited attempts on learning low-level locomotion control from offline datasets. Most of the prior works focus on learning simple tasks within simulation, such as Gym locomotion tasks, mainly with offline RL [34]. Among them, some leverage Q-learning on offline datasets [43, 32, 31], while others utilize supervised learning conditioned on rewards [64, 11, 69]. However, these tasks are oversimplified and do not adequately consider the complexities encountered in real-world scenarios. The efficacy of offline RL in controlling physical robots remains unproven. An alternative is the use of offline data as a foundation for online learning [42, 60]. Among them, Smith et al. develop baseline policies from offline datasets to bootstrap online learning. Yet, this approach still requires online learning, which again introduces challenges in learning multi-modal policy. In comparison, in this work, we develop an offline learned policy directly deployable in the real world without online learning, leading to a more effective multi-skill policy and simpler training scheme.

1.2.3 Diffusion Models

Diffusion models, particularly denoising diffusion probabilistic models (DDPM) [28], represent a class of generative models in machine learning that have gained significant attention for their ability to generate high-quality, detailed samples, such as images, audio, and text. These models operate on the principle of gradually adding noise to data and then learning to reverse this process, effectively 'denoising' to recreate the original data.

The core idea behind diffusion models, inspired by the physical process of diffusion, is to model the data generation process as a Markov chain of gradual, random steps of adding and removing noise. In the context of DDPM, the forward process incrementally adds Gaussian noise to the data over a sequence of steps, transforming the data distribution into a known noise distribution. The reverse process, which is the generative phase, aims to learn the reverse transition of this Markov chain, starting from noise and progressively denoising it to generate data samples.

1.2.4 Diffusion Models in Robotics

Recent advances have seen an increasing number of applications of diffusion models for learning control and planning systems. Many prior works have focused on integrating diffusion as part of the autonomy pipeline. Wang et al. leverage a diffusion model as the discriminator

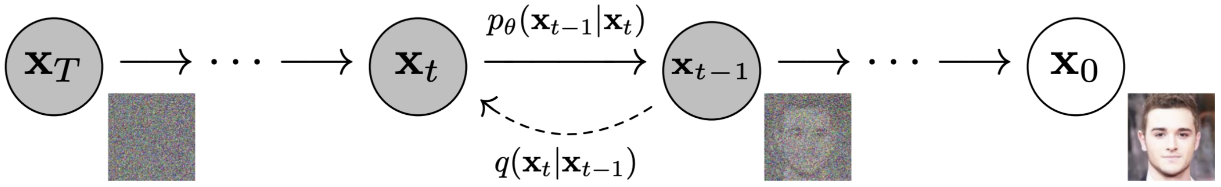
$\mathbf{x}_T$
$\mathbf{x}_t$
$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$
$\mathbf{x}_{t-1}$
$q(\mathbf{x}_t|\mathbf{x}_{t-1})$
$\mathbf{x}_0$

Figure 1.3: Diagram of the Denosing Diffusion Probabilistic Models(DDPM) Process[28]

in adversarial IL for legged control[62]. Nuti, Franzmeyer, and Henriques and Wang, Chen, and Sun learn a reward model with diffusion to train a control policy with RL. However, the actor networks in these works are small multi-layer perceptron (MLP) networks, which still suffer from the problem of exploration difficulty in multimodal learning [64]. Others leverage diffusion models in high-level planning to perform trajectory planning [27, 23], to extend towards safe planning [68], or to generate goals for low-level inverse-kinematics-based controller [29, 1]. Many works have been particularly instrumental in enhancing visuomotor planning for manipulation tasks, such as [47, 51, 40]. However, the aforementioned methods all require low-level controllers to realize high-level diffusion-based planning, and are limited to simulation.

Among work that attempts to solve manipulation tasks by diffusion, notable efforts extend beyond simulation to real-world robots. Diffusion policy [15] works on leveraging diffusion to perform a wide variety of manipulation tasks on robots with visual inputs. Li et al. improve it by adding self-supervised learning in crossway diffusion. Some works further integrate language as conditioning [10, 21]. Yoneda et al. leverages reverse diffusion process for shared autonomy with human user in end-effector planning. Other proposals include hierarchical frameworks to decompose tasks that require multiple skills [3, 67] and Octo Model Team push it further to form a generalist policy from large dataset of various source policies.

However, all of the abovementioned approaches focus on high-level planning on manipulation systems, featuring a low-dimensional action space (*e.g.*, end-effector position), low-replanning frequency (*e.g.*, around 10 Hz), and inherently more stable dynamics.

In contrast, there are limited attempts of using diffusion models in low-level control policies for high-frequency control. Most relevant work like [70] uses online RL to train an actor policy represented by a diffusion model in simple simulation environments (*e.g.*, Gym control tasks). However, unlike simulation environments that require less robustness and assume privileged information, real-world legged robots necessitate high-frequency feedback control due to their instability and rapid dynamics [65]. This introduced significant difficulty in leveraging diffusion models which are usually represented by a large model [8]. As we will see, this work does not only realize diffusion models in the low-level control for legged locomotion but also demonstrates their advantages in the control tasks in the real world.

Chapter 2

Method

2.1 Overview

In this section, we provide an overview of *DiffuseLoco*, a framework designed to generate and utilize offline datasets for training locomotion policies. DiffuseLoco is based on diffusion models and is designed to train a low-level multi-skill locomotion policy from offline datasets distilling diverse behaviors.

A schematic illustration of the three stages of the DiffuseLoco framework is shown in Figure 2.2.

2.2 Data Sources①

We start with collecting a diverse dataset consisting of multiple skills. We first obtain N single-skill control policies as the source policies. In this work, the source policies are pacing, trotting, turning left, and turning right skill-specific policies, as illustrated in Fig. 2.2, *i.e.*, $N = 4$. These policies are obtained by RL [48] and are conditioned on given goal $\mathbf{g}$ (command). For example, the robot can use the pacing policy to track different speeds (goals). We then collect data generated by each of these goal-conditioned source policies in simulation. As illustrated in Algo. 1, during the data collection, we start an episode where the robot is controlled by the i^{th} source policy π^i. The state-action-goal pairs ($\mathbf{s}_t$, $\mathbf{a}_t$, $\mathbf{g}_t$) are collected during the rollout of the robot's closed-loop dynamics until it reaches the maximum episode length T. Specifically, the goal $\mathbf{g}_t$ will be re-sampled within the command range after a time-interval within the episode, and the states $\mathbf{s}_t$ and action $\mathbf{a}_t$ are the proprioceptive feedback from the robot and the joint-level commands from the source policy, respectively. We repeat such a process over all of the N source policies and collected an offline dataset that contains 1 million ($\mathbf{s}_t$, $\mathbf{a}_t$, $\mathbf{g}_t$) pairs in total.

① I have collaborated with Xiaoyu Huang in the section. He was in charge of the data source collection and partially the training process.

2.3 Training

In the second step of the framework, we train our DiffuseLoco policy from the generated dataset in an end-to-end manner. Let input state and goal history length be h and output action prediction length be n. During training, we sample a segment of state trajectory $\mathbf{s}_{\text{traj}}$ and corresponding action and goal sequences, $\mathbf{a}_{\text{traj}}$ and $\mathbf{g}_{\text{traj}}$. We sample a diffusion timestep k randomly from $\{1, \ldots, K\}$, and sample a Gaussian noise ϵ_k to add to the action sequence. Then, a transformer-based denoising model takes the noisy action sequence along with states trajectory $\mathbf{s}_{\text{traj}}$, goal trajectory $\mathbf{g}_{\text{traj}}$, and diffusion timestep k as input, and predicts the added noise as ϵ_θ. The predicted noise ϵ_θ is then regressed to match the true noise ϵ_k with mean square error loss. In this way, the denoising model is learned to generate sequences of low-level actions conditioned on robot states and goals from the dataset.

2.4 Deployment

In the last stage of the framework, we zero-shot transfer the trained DiffuseLoco policy on the robot hardware. The policy is designed to operate at a frequency of 30 Hz. During deployment, the DiffuseLoco policy takes a sequence of noisy actions sampled from a Gaussian distribution, and denoises it conditioned on the state trajectory $\mathbf{s}_{\text{traj}}$ from the robot hardware and the given goal $\mathbf{g}_{\text{traj}}$. The denoising process is repeated for K iterations to generate a sequence of action, but only the *immediate* action $\mathbf{a}_t$ is taken for the robot's joint-level command. After executing the actions, the DiffuseLoco policy takes a new sequence of states from the robot and updates the action from the newly-generated action sequence. This is designed to align with the Receding Horizon Control (RHC) framework, instead of interpolating the action sequence at high frequency as previously used by other diffusion-based work [29, 35]. However, since the diffusion model has a large number of parameters, we need to accelerate the inference fast enough to achieve this RHC manner (the inference time should be faster than the control frequency of 30 Hz).

Using this framework, we are able to obtain and utilize DiffuseLoco policy on the physical robot. As presented in Sec. 4.2, DiffuseLoco policy is able to control the robot to walk stably and track varying velocity commands as a goal. More importantly, a single DiffuseLoco policy can perform multiple skills learned from the offline dataset and infer smooth skill transitions which are not included in the dataset.

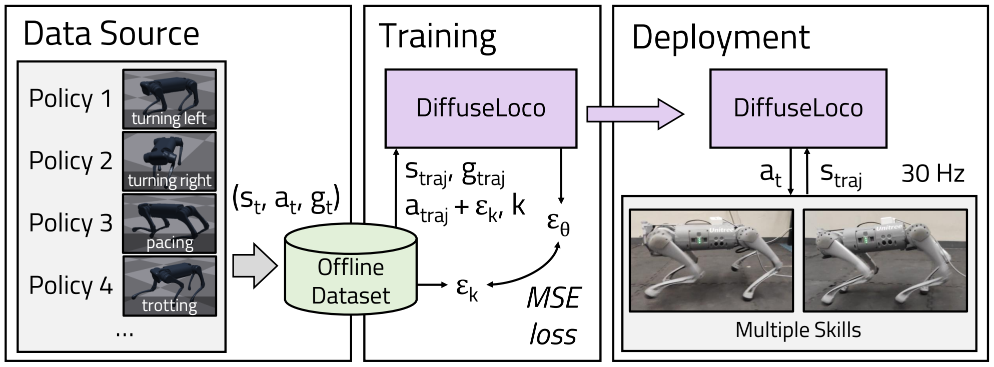

Figure 2.2: Overview of the three stages of DiffuseLoco. First, we generate an offline dataset with a set of skill-specific source policies (left). Then, we train DiffuseLoco policy with DDPM loss on trajectories within the dataset (middle). Finally, DiffuseLoco policy is deployed on the robot in the real world and executes multiple skills (right).

2.5 Diffusion Model for Real-Time Control

Having introduced the framework of DiffuseLoco, we now begin to develop the backbone of this framework: a diffusion model for locomotion control, shown in Fig. 2.3. During development, we pay a special focus on design choices for real-time control and inference acceleration.

2.5.1 DDPM for Control

In order to model multi-modal behaviors from diverse datasets, we leverage DDPM [22] to model different skills that can be applied to achieve a common goal. DDPM is a class of generative models in which the generative process is modeled as a denoising procedure, often referred to as Stochastic Langevin Dynamics. To generate the action trajectory for control, an initial noisy action trajectory, $\mathbf{a}^K_{t:t+n}$, is sampled from Gaussian noise, and the DDPM conditioned on state trajectory $\mathbf{s}_{t-h:t}$ and previous action trajectory $\mathbf{a}_{t-h-1:t-1}$ undergoes K iterations of denoising. For clarity, from now, the subscript $*_{t-a:t-b}$ is used to detail the trajectory from timestep $t-a$ to $t-b$ to replace the previously-used $*_{\text{traj}}$. Such a denoising process yields a sequence of intermediate actions characterized by progressively decreasing noise levels: $\mathbf{a}^k, \mathbf{a}^{k-1}, \ldots, \mathbf{a}^0$, until the desired noise-free output, $\mathbf{a}^0$, is attained. This process can be expressed as the following equation:

$$\begin{aligned}\mathbf{a}^{k-1}_{t:t+n} = \alpha(\mathbf{a}^k_{t:t+n} - \gamma\epsilon_\theta(\mathbf{a}^k_{t-h-1:t+n}, \mathbf{s}_{t-h:t}, k) \\ + \mathcal{N}(0, \sigma^2 I))\end{aligned} \tag{2.1}$$

where $\mathbf{a}^k_{t:t+n}$ represents the output at the k^{th} iteration, and $\epsilon_\theta(\mathbf{a}^k_{t-h-1:t+n}, \mathbf{s}_{t-h:t}, k)$ represents the predicted noise from the denoising model ϵ_θ, which is parameterized by θ, with respect

Algorithm 1 DiffuseLoco Algorithm

```
1: Initilize: N source policies $\pi^1_{src} \ldots, \pi^N_{src}$, Empty Offline Dataset $\mathcal{D}_{src}$, Diffusion Model $\pi_\theta$
2: // OBTAIN DATA FROM $\pi_{src}$
3: if $\mathcal{D}_{src}$ is empty then
4:     repeat
5:         Sample $n$ uniformly from $\{1, \ldots, N\}$
6:         Sample environment dynamics
7:         for $t = 1$ to $T$ do
8:             if $t \mod \text{random\_goal\_step} = 0$ then
9:                 Sample goal $\mathbf{g}$
10:            end if
11:            Collect data with $\pi^n_{src}(\mathbf{a}_t|\mathbf{s}_t, \mathbf{g}_t)$
12:            Add data to offline dataset $\mathcal{D}_{src} \leftarrow (\mathbf{s}_t, \mathbf{a}_t, \mathbf{g}_t)$
13:        end for
14:    until desired
15: end if
16: // TRAIN ON OFFLINE DATASET
17: for each epoch do
18:     for each $(\mathbf{s}_{\text{traj}}, \mathbf{a}_{\text{traj}}, \mathbf{g}_{\text{traj}})$ in $\mathcal{D}_{src}$ do
19:         Compute the loss $L(\theta)$
20:         Update model parameters $\theta$ to minimize loss $L(\theta)$
21:     end for
22: end for
```

to $\mathbf{a}^k_{t-h-1:t+n}$, $\mathbf{s}_{t-h-1:t}$, and iteration k. $\mathcal{N}(0, \sigma^2 I)$ denotes the sampled noise from a DDPM scheduler. This scheduler takes in α, γ, and σ as its hyperparameters, where α regulates the rate at which noise is added at each step, γ represents the denoising strength at each step, and σ defines the noise level.

During training, we opt to use the simplified training objective as proposed in [22],

$$l = MSE(\epsilon_k, \epsilon_\theta(\mathbf{a}_{t-h-1:t+n} + \epsilon_k, \mathbf{s}_{t-h:t}, k)) \tag{2.2}$$

where ϵ_k is the sampled noise at iteration k.

2.5.2 Robot's I/O History as Input

We now develop the denoising model based on a transformer architecture, as shown in Fig. 2.3. The input to the denoising model contains a history of previous actions $\mathbf{a}_{t-h-1:t-1}$ along with robot state $\mathbf{s}_{t-h:t}$, *i.e.*, robot's I/O history as input. The state $\mathbf{s}_t$ and action $\mathbf{a}_{t-1}$ pair at each timestep is concatenated into one vector, the I/O vector. The robot's I/O history helps the policy to better perform system identification and state estimations in legged locomotion control, as evaluated in [37].

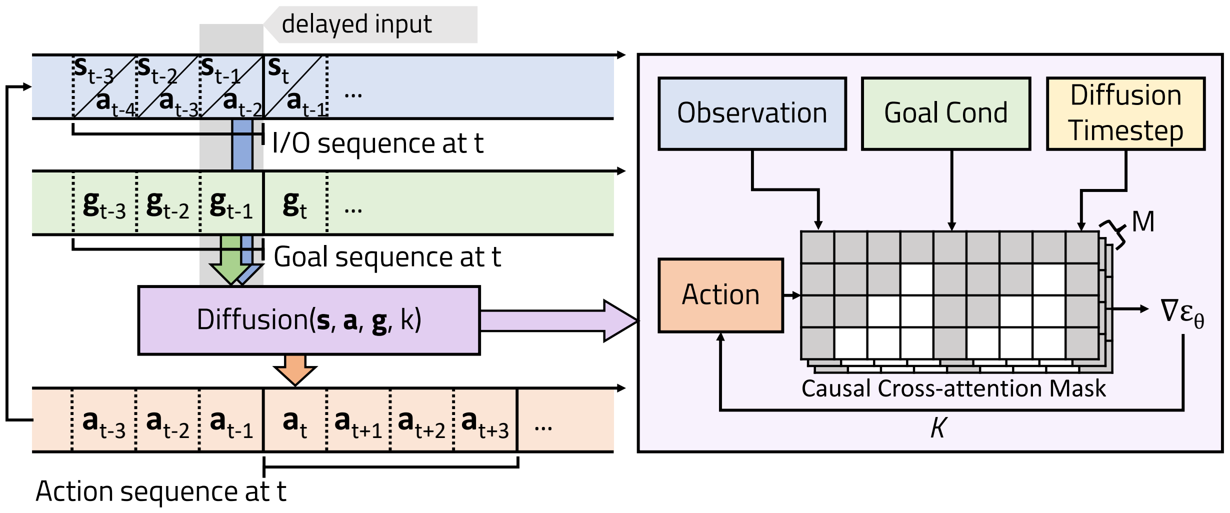
delayed input
s_{t-3} a_{t-4} s_{t-2} a_{t-3} s_{t-1} a_{t-2} s_t a_{t-1} ...
I/O sequence at t
g_{t-3} g_{t-2} g_{t-1} g_t ...
Goal sequence at t
Diffusion(s, a, g, k)
a_{t-3} a_{t-2} a_{t-1} a_t a_{t+1} a_{t+2} a_{t+3} ...
Action sequence at t
Observation
Goal Cond
Diffusion Timestep
M
Action
∇ε_θ
Causal Cross-attention Mask
K

Figure 2.3: The architecture. At time step t, takes in a one-step delayed history of proprioceptive state $\mathbf{s}_{t-h-1:t-1}$, along with the corresponding delayed goal history $\mathbf{g}_{t-h-1:t-1}$ and action history $\mathbf{a}_{t-h-2:t-2}$, and predicts a sequence of n actions $\mathbf{a}_{t:t+n}$ as the position targets for robot's actuators. First, the proprioceptive state and the goal are mapped to separate embedding spaces by MLP encoders. Next, the noisy action tokens undergo M transformer decoder layers, within which the action tokens query both the state embedding and goal embedding, as well as a ont-hot diffusion timestep, through a causal cross-attention module. Finally, we repeat the denoising process K times to produce the predicted action sequence, and feed the executed actions back to model's input. The model is trained through end-to-end imitation learning.

2.5.3 Goal Conditioning

Besides the robot's I/O history, the input to the DiffuseLoco policy has an additional formulation of the given goal. Different from prior works which concatenate state and goal into the same embedding [15, 35], in DiffuseLoco, we propose adding goal conditioning to the DiffuseLoco policy, with a special formulation. This approach recognizes the difference in features of the state-action pair (robot's I/O) and goal spaces, and separates the embedding of these two information. Such separation between robot's I/O and goal is crucial for real-time dynamic control scenarios. Using legged robots as an example, unlike static-base manipulators, the state $\mathbf{s}_{t-h:t}$ (the proprioceptive readings of the robot) and used action can change rapidly, while the goals $\mathbf{g}_{t-h:t}$ (commands) of the control policy largely stays static. Therefore, there would be different priorities when regarding the robot's I/O and goal. For example, in the event where the robot encounters environment changes or external perturbations, it needs to focus on the robot's I/O history to capture the changing dynamics and prioritize the rebalancing of the body. On the other hand, the policy can focus more on realizing the given goal when the body stability is not a big concern.

Therefore, we separate the two spaces of the robot's I/O and goal information into separate conditioning embeddings and calculate individual attention weights when doing cross-attention with the action embedding, as illustrated in Fig. 2.3. Concretely, we modify Eqn. 2.1 and introduce the goal embedding $\mathbf{g}$ as follows:

$$\begin{aligned}\mathbf{a}^{k-1}_{t:t+n} = \alpha(\mathbf{a}^{k}_{t:t+n} - \gamma\epsilon_\theta(\mathbf{a}^{k}_{t-h-1:t+n}, \mathbf{s}_{t-h:t}, \mathbf{g}_{t-h:t}, k) \\ + \mathcal{N}(0, \sigma^2 I)).\end{aligned} \tag{2.3}$$

We find that this enables better command tracking performance and achieves better robustness compared to the policy without such condition in the legged locomotion control. A detailed ablation study is conducted in Sec. 4.3.3.

2.5.4 Model Architecture and Training Details

After introducing all the design choices in the DiffuseLoco policy for real-time control, we now present the details of the architecture of diffusion model developed in this work. The DiffuseLoco policy leverages an encoder-decoder transformer DDPM, incorporating the aforementioned goal conditioning, as illustrated in Fig. 2.3.

First, the past robot's past I/O trajectory $(\mathbf{s}_{t-h-1:t-1}, \mathbf{a}_{t-h-2:t-2})$ and given goal sequence $\mathbf{g}_{t-h-1:t-1}$ are transformed into separate I/O embedding and goal embedding by two 2-layer MLP encoders, respectively. Then, we sample noise $\epsilon(k)$ for diffusion time step k with the DDPM scheduler and add to the ground truth action $\mathbf{a}$ from the offline dataset to produce a noisy action $\mathbf{a}^k_{t:t+n} = \mathbf{a}_{t:t+n} + \epsilon_k$. The noisy action $\mathbf{a}^k_{t:t+n}$ is then passed through an MLP layer into action embedding. The noisy action tokens are then passed through 6 Transformer decoder layers, each of which is composed of an 8-head cross-attention layer. Each layer computes the attention weights for the noisy action tokens querying all the state embedding, goal embedding, and the timestep embedding reflecting the current diffusion timestep k. We apply causal attention masks to each of the state embeddings and goal embeddings separately. The predicted noise $\epsilon_\theta(\mathbf{a}_{t-h-2:t+n}, \mathbf{s}_{t-h-1:t-1}, \mathbf{g}_{t-h-1:t-1}, k)$ is then computed by each corresponding output token of the decoder stack. We then supervise the output to predict the added noise with the loss function to find optimal parameters θ of the denoising model ϵ_θ.

For quadrupedal robots, we select following hyperparameters: state and goal history length $h = 8$, inference horizon $n = 4$, and diffusion inference steps $K = 10$.

Chapter 3

Real-Time Inference Acceleration for DiffuseLoco Policy[②]

Although the diffusion model targets real-time use, it cannot meet the real-time targets without further tuning. Compared to previous works that use Transformers for locomotion control with 2M parameters [50], our model is 3 times larger in parameter counts(6.8M parameters) and needs to be forwarded 10 times in each inference. Thus, an additional effort is needed to accelerate the diffusion on the edge computing device on the robot, such as the setup shown in Fig. 3.1. In this section, we explore several methods to accelerate the inference process of the diffusion model to enable it to run real-time onboard.

3.1 Acceleration Framework

Our DiffuseLoco policy has a parameter count of 6.8 million parameters, which exceeds most modern mobile processors' cache capacity. Furthermore, hardware on a typical consumer-grade central processing unit (CPU) is not optimized for the operators used in transformer networks. The graphics processing unit (GPU) is more suitable for computing the high-dimension matrix and vector operations. To ensure the portability of the setup, we use an accessible NVIDIA GeForce RTX 4060 Mobile GPU as the deployment platform. For real-time deployment, an acceleration pipeline is built in the DiffuseLoco framework to convert and optimize our model towards the target compute platforms. The operators of the model are first extracted with ONNX [59]. Then, TensorRT is used to refine the execution graph and compile the resulting execution pipeline onto the target GPU. Through domain-specific architecture optimizations, the operations and memory access patterns are optimized to utilize the full capability of the GPU. With this approach, the time for each denoising iteration is decreased by about 7 times compared to the native implementation in PyTorch, and the maximum inference (with 10 denoising iterations) frequency is increased from 17.0 Hz to 116.5 Hz. To showcase the effect of this acceleration approach, we conducted a benchmark on the inference frequency of the policy running on multiple hardware platforms we have

② The deployment and benchmarking sections were done in collaboration with Yufeng Chi. We set up the onboard compute system and acceleration pipeline together.

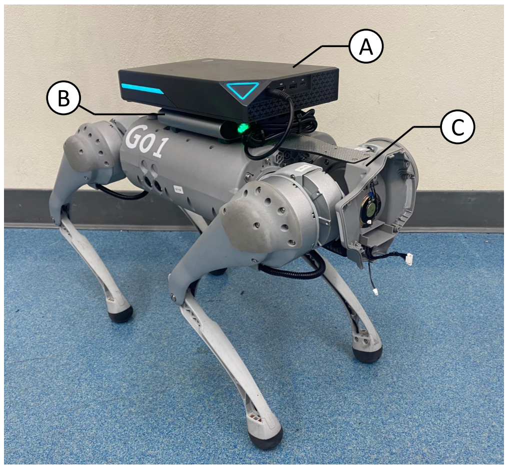

Figure 3.1: Onboard compute experiment setup. A: Mini computer with Intel Core i7-13700H and NVIDIA GeForce RTX 4060 Mobile. B: Battery bank. C: Go1 quadrupedal robot. This setup is below the robot's adaptive load capacity and the robot can walk with our policy steadily. With our acceleration framework, we are able to achieve onboard and real-time deployment of our diffusion model.

access to, shown in Fig. 3.2.

3.2 Edge Compute for DiffuseLoco Policy on Robots

With the help of the acceleration framework, the compute platform can be deployed onboard a Go1 quadruped robot. A mini-computer equipped with Intel Core i7-13700H and NVIDIA GeForce RTX 4060 Mobile is attached to the top of the robot, as showcased in Fig. 3.1. This computer runs DiffuseLoco policy and is powered by a dedicated battery bank, separated from the robot's internal battery. This arrangement is capable of running the policy for up to 90 minutes. The mini-computer connects to the robot via an Ethernet cable to send action for the joint-level PD controls on the robot's computer.

To this end, the development of the entire DiffuseLoco framework has been presented, where we developed a policy that is able to *zero-shot transfer* onto real robots, track *different velocity commands*, and execute *multiple skills* to perform stable locomotion. In the following sections, we conduct extensive benchmark and ablation study to evaluate DiffuseLoco in both simulation and the real world.

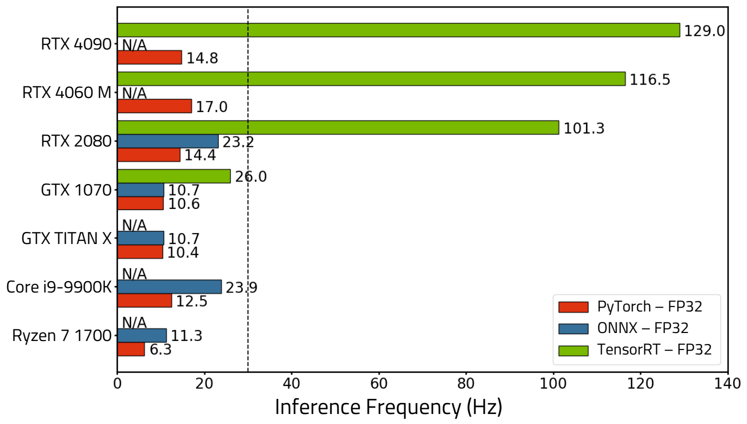

Figure 3.2: Benchmark of running our DiffuseLoco policy (6.8M parameters; about 27.2MB) on different hardware platforms. The dashed line remarks the 30 Hz minimum frequency required to control the robot in real time. We utilize TensorRT to optimize the computation graph and achieve approximately 7 times speedup of inference computation time compared to the naive PyTorch implementation. The N/A entries are due to the lack of software compatibility on the corresponding platform.

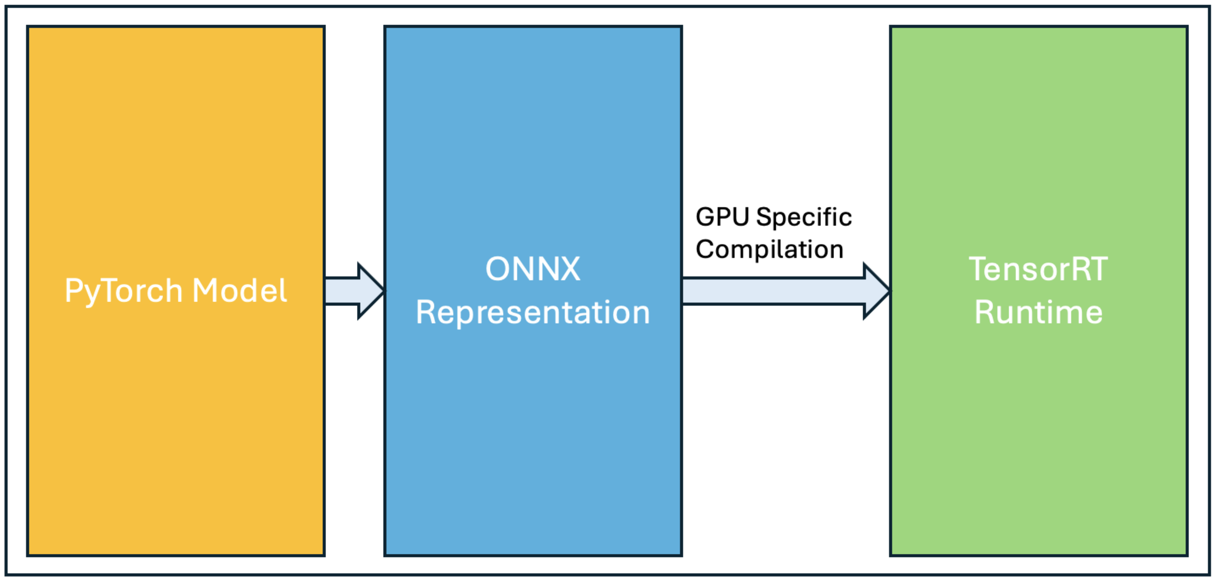

Figure 3.3: Full stack diagram of acceleration pipeline. Our pipeline first convert Pytorch Model to ONNX representation, then we build the TensorRT runtime with specific GPU.

Chapter 4

Real-World Experiments and Ablation Studies③

In order to demonstrate our framework can learn multiple skills from various sources, we run a comprehensive real-world benchmark.

4.1 Experiment Setup

In this section, we discuss the benchmark tests and evaluation metrics conducted in the subsequent sections on a physical quadrupedal robot.

The test includes walking on a foam-padded floor for four meters under five goals (commands) with different velocities. The goals are the following: move forward at three different speeds: 0.3 m/s, 0.5 m/s, and 0.7 m/s, and make a left turn and a right turn at 0.3 rad/s. These commands represent commonly-used velocities in quadruped's locomotion and are included in the training dataset.

For evaluating the performance, we examine the *stability* and velocity *tracking performance* produced by the controllers. The metric for *stability* is assessed by calculating the average number of trials in which the robot manages to move stably for a minimum of 5 seconds without falling over. The velocity *tracking error* is reported as the mean and standard deviation of the error between the commanded velocity and the robot's actual velocity, expressed as a percentage of the commanded velocity. Each test is repeated five times in a continuous sequence without interruption. Additionally, we record the norm of angular velocity in roll and pitch directions, as another perspective to measure the smoothness of the resulting robot's locomotion skills.

4.2 Experiment: Real-world Benchmarks

In this section, we benchmark **DiffuseLoco** against state-of-the-art methods that leverage RL for multi-skill locomotion control using a single policy and variants of behavior cloning

③ All experiments were conducted in real-world settings with Yufeng and Xiaoyu's assistance. Together, we set up and monitored over 200 experimental setups, ensuring accurate recordings.

Task	**Metric**	**AMP**	**AMP w/ H**	**TF**	**TF w/ RHC**	**DiffuseLoco (Ours)**
$0.3m/s$ **Forward**	Stability (%)	**100**	**100**	80	**100**	**100**
	E_v (%)	90.44 ± 1.87	90.63 ± 4.79	75.75 ± 6.07	39.28 ± 2.34	**33.22** ± **12.48**
$0.5m/s$ **Forward**	Stability (%)	**100**	**100**	**100**	**100**	**100**
	E_v (%)	50.44 ± 1.97	46.29 ± 2.55	54.35 ± 2.66	37.46 ± 5.31	**12.91** ± **6.84**
$0.7m/s$ **Forward**	Stability (%)	0	20	0	40	**100**
	E_v (%)	~~fail 5/5~~	54.96 ± 0.00	~~fail 5/5~~	39.36 ± 5.02	**24.80** ± **8.91**
Turn Left	Stability (%)	20	**100**	0.	**100**	**100**
	E_v (%)	20.96 ± 0.00	33.39 ± 6.96	~~fail 5/5~~	13.41 ± 5.02	**12.79** ± **5.64**
Turn Right	Stability (%)	**100**	**100**	**100**	80	**100**
	E_v (%)	18.61 ± 2.40	33.39 ± 6.96	25.86 ± 1.47	8.69 ± 5.04	**2.22** ± **1.03**

Table 4.1: Performance benchmarks across different baselines and our DiffuseLoco policy in the real world. Stability (the higher the better) measures the number of trials in which the robot stays stable and does not fall over. E_v (the lower the better) measures the deviation from the desired velocity in percentage. The experiments are conducted with different command settings (Task). Each command is repeated non-stop for five trials, and we report the average and standard deviation of the metrics across five trials.

(BC) baselines on the Go1 quadrupedal robot *in the real world*. Our experiments show that **DiffuseLoco** can zero-shot transfer to hardware and exhibits better stability and control performance (*i.e.*, velocity tracking error) compared to all compared baselines.

4.2.1 Baselines

First, we compare our policy with state-of-the-art multi-skill RL policies *without explicit skill conditioning*. We deem this a favored property for a versatile offline learning framework, as skill labels might not be always available during deployment. In addition, our **DiffuseLoco** policy is purely goal-conditioned.

- Adversarial Motion Priors (**AMP**) [18]: An MLP policy trained using AMP with RL (PPO) and the style reward is obtained by the discriminator from *different* reference motions. We use the open-sourced checkpoint from [18]. This resulted single RL policy can perform four different skills (pacing, trotting, turning left and right) as ours in simulation. Note that in the implementation of [18], the AMP-based policy does not include a history of states as input.
- AMP with history steps (**AMP w/ H**): To align with **DiffuseLoco**, we in addition train an MLP policy with 8 steps of state and action history using AMP and the same

setup as [18]. This baseline achieved a similar evaluation return as the **AMP** baseline ([18]) in simulation.

Although the model is trained on A1(previous version of Go1), we find that Go1 yields very similar performance as the two robots are mostly designed the same. For consistency, we report performance on Go1 for all policies.

Furthermore, we compare **DiffuseLoco** with the following BC policies. BC policies can be generally placed into two categories: autoregressive token prediction [11, 25] and action sequence prediction as used in [20]. We adopt baselines for each category.

- Transformer with Autoregressive Token Prediction (**TF**): A Generative Pretrained Transformer (GPT) [6] policy similar to a decision transformer [11] without reward conditioning. This only generates one timestep action.
- Transformer with Receding Horizon Control (**TF w/ RHC**): A transformer policy with the same future step action predictions. The model's architecture is identical to our **DiffuseLoco** model, but it directly predicts future action sequences and the loss is replaced by the BC loss $l = MSE(\pi_\theta(\mathbf{s}_t, \mathbf{g}_t), \mathbf{a}_t)$.

All BC baselines have the same parameter count of 6.8M and are trained with the same learning rate scheme and number of epochs as **DiffuseLoco**.

For the same reasons as above, we do not include rewards in the dataset and offline RL methods.

Typically, previous work uses DAgger style algorithms [53] to better cope with distribution shift, but these methods require access to the expert policy and online learning environment. As a more versatile framework, we limit our focus on learning from offline datasets only.

4.2.2 DiffuseLoco versus AMP (RL)

We first compare DiffuseLoco with RL-based multi-skill control policy **AMP**. Table 4.1 shows that **DiffuseLoco** is the only method among all of the baselines in our benchmark that is able to reliably complete all trials without falling over. Specifically, the RL-trained **AMP** and **AMP w/ H** baselines struggle with low and high speed commands. For 0.3 m/s forward command, these two baselines give a velocity tracking performance of more than 90% slower than the commanded velocity. For 0.7 m/s forward command, they achieve a stability metric of 0% and 20% respectively.

This shows the prevailing problem of mode-collapsing on Generative Adversarial Network (GAN) style networks, such as a multi-skill AMP policy. Mode collapse is a significant challenge in GANs, where the generator becomes overfitted to a limited range of outputs that are often similar or identical, rather than offering a broad range of solutions. This issue typically arises when the generator produces a set of good samples. The sample traps the discriminator in a local optimum, which in turn makes the generator overfit to these

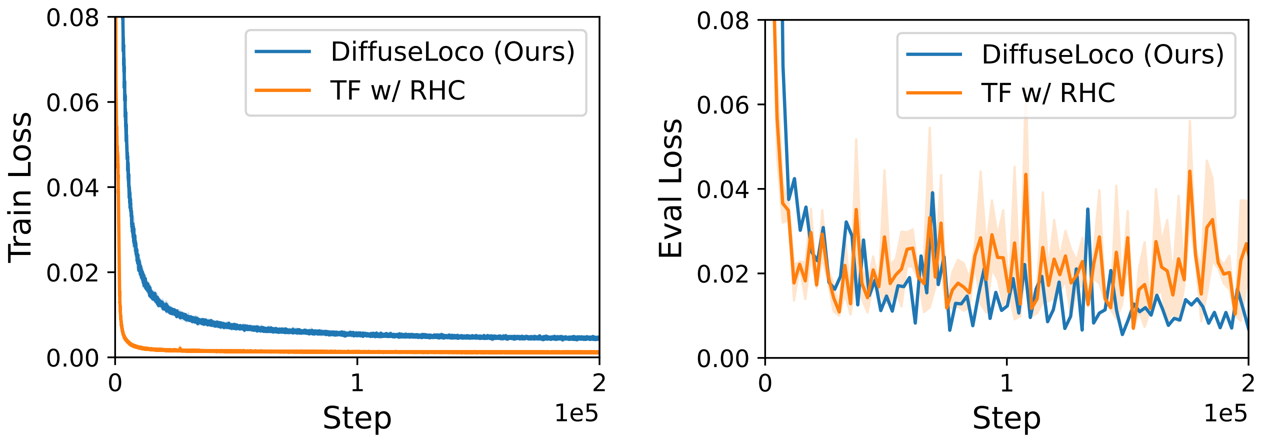

Figure 4.1: Loss curves for training **DiffseLoco** and **TF w/ RHC** baseline. (a): Training Loss. (b): Evaluation Loss. We report mean and standard deviation across three seeds. We find that even though **TF w/ RHC** achieves a low reconstruction loss in training, the evaluation loss stays higher than **DiffseLoco** and increases at the end. This indicates **TF w/ RHC** tends to overfit the offline datasets while *DiffuseLoco* is not, with the same number of parameters.

samples [41, 38, 17]. In the context of AMP, this means the actor network excessively overfits to the simulation environment, losing its ability to generalize and adapt to new environments (such as the real world). Although there are work that try to fix this problem with GAN network, they require extensive conditionals with extra information.[12] In real-world testing, this results in the policy hobbling and almost staying still when speed is low, and cannot exercise a balanced locomotion skill when speed is high. Note that in the simulation environment, both **AMP** and **AMP w/ H** are able to control the robot to track different velocities without falling over.

Besides better sim-to-real transfer, **DiffuseLoco** with a diffusion model is able to efficiently learn the multi-modality presented by the different skills for the same locomotion task, and thus able to perform valid and coordinated locomotion skills without mode-collapsing, as an example given by Fig. 4.3. This helps **DiffuseLoco** achieve both better stability and track completion (velocity tracking) performance compared with AMP-based policy baselines.

4.2.3 DiffuseLoco versus Behavior Cloning

We further compare **DiffuseLoco** with BC-based methods. For locomotion tasks, smooth and temporally consistent actions are a necessity for stability and robustness. Looking at Table 4.1, we find that **DiffuseLoco** outperforms **TF** and **TF w/ RHC** in both stability and robustness of the locomotion policy. Using one-step action output, we find that **TF** lacks robustness and fails the 0.7 m/s forward and left turn tasks completely. This is because single-step action prediction lacks consistency and is likely to produce rapidly changing actions (resulting in jittering motion) when inferring in an autoregressive manner.

With receding horizon control, **TF w/ RHC** overcomes most of the jittering problem and can complete most of the tasks. However, we note that for more agile motion such as the 0.7 m/s forward task, the stability metric drops drastically to merely 40%. This is likely because the BC loss used in **TF w/ RHC** training tends to overfit the action trajectories in the dataset, resulting in less robust policy in the out-of-distribution scenarios (such as in the real world).

This is especially evident when looking at training curves for **TF w/ RHC** versus **DiffuseLoco** shown in Figure 4.1, where **TF w/ RHC** overfits significantly to the training dataset and the evaluation loss stays high. Note that the model architecture is kept identical across **TF w/ RHC** and **DiffuseLoco**, so only the loss calculation is changed in this comparison. In addition, the evaluation loss here is calculated with samples from the same distribution as the training dataset, which means the overfitting problem becomes more pronounced when we switch to real-world experiments, as shown earlier.

In comparison, our **DiffuseLoco** shows more stable and smooth motions measured by both stability metrics and magnitudes of body's angular velocity. On average, **DiffuseLoco** achieves 10.40% less in magnitude for body's oscillation over all trials. As a result, the smoother locomotion skill helps **DiffuseLoco** to achieve on average 38.97% less tracking error compared to **TF w/ RHC**. Based on this observation, we suggest that DDPM style training is more suitable for imitating locomotion tasks compared to vanilla Behavior Cloning.

4.2.4 Summary of Results

In summary, after the benchmark with baselines, we can draw the following conclusions. Compared with RL-based multi-skill locomotion control policies (AMP-based), our **Dfis-sueLoco** shows better control performance during sim-to-real transfer as it does not suffer from the potential mode-collapsing issue encountered by GAN-styled methods. This result is consistent compared with AMP-based policies that uses only 1-timestep robot's I/O as input (**AMP**) or uses a history of the robot's I/O (**AMP w/ H**). Compared with BC-based policies, our **DfiffuseLoco** that leverages diffusion shows less tendency to overfit the offline dataset and results in better sim-to-real transfer performance in terms of stability and tracking performance.

4.3 Experiment: Ablation Study on Design Choices

In this section, we further evaluate the design choices used to build DiffuseLoco policy in simulation and the real world by extensive ablation studies.

Task	Metric	DL w/o RHC	DL w/o Rand	DDIM-100/10	DDIM-10/5	DiffuseLoco (Ours)
$0.3m/s$ **Forward**	Stability (%)	**100**	**100**	**100**	**100**	**100**
	E_v (%)	75.09 ± 18.98	50.45 ± 2.70	56.89 ± 2.43	47.09 ± 2.40	**33.22 ± 12.48**
$0.5m/s$ **Forward**	Stability (%)	**100**	80	80	**100**	**100**
	E_v (%)	64.49 ± 1.87	41.07 ± 6.12	41.00 ± 3.18	37.92 ± 1.59	**12.91 ± 6.84**
$0.7m/s$ **Forward**	Stability (%)	0	40	80	80	**100**
	E_v (%)	~~fail 5/5~~	44.30 ± 4.21	47.71 ± 6.63	42.58 ± 2.08	**24.80 ± 8.91**
Turn Left	Stability (%)	**100**	**100**	**100**	**100**	**100**
	E_v (%)	20.96 ± 18.22	**10.17 ± 5.86**	22.22 ± 4.29	13.27 ± 2.63	12.79 ± 5.64
Turn Right	Stability (%)	**100**	**100**	**100**	**100**	**100**
	E_v (%)	18.61 ± 2.40	8.18 ± 3.94	6.47 ± 2.49	7.42 ± 2.90	**2.22 ± 1.03**

Table 4.2: Performance Ablation Study across different ablations and DiffuseLoco policy in real-world experiments. Stability (the higher the better) measures the number of trials in which the robot stays stable and does not fall over. E_v (the lower the better) measures the deviation from the desired velocity in percentage. The experiments are conducted with different command settings (Left). Each command is repeated non-stop for five trials, and we report the average and standard deviation of the metrics across five trials.

4.3.1 Ablation Components

To validate our design choices, we define **DiffuseLoco** with the following critical components and compare in our real-world benchmark.

- Without Receding Horizon Control (**DL w/o RHC**) [11]: In this baseline, we replace receding horizon control with one-step prediction in an autoregressive manner and keep the diffusion model.
- Without Goal-conditioning (**DL w/o Goal**): In this baseline, we do not add the goal-conditioning encoder. Instead, the goal is concatenated with the robot's I/O.
- Without Domain Randomization (**DL w/o Rand**): In this baseline, we investigate how the diversity of the dataset may influence the robustness of **DiffseLoco**. During the date generation of the offline dataset, we disable all domain randomization except for the ground friction coefficient.
- DDIM Inference: DDIM uses less number of denoising steps at inference than the one used in training, which is a commnely-used technique to accelerate diffusion models at deployment. This is in contrast to our proposed DiffuseLoco that keeps the same number of denoising steps with DDPM sampler. We developed two baselines based on DDIM to investigate how training and inference steps affect performance in locomotion control.

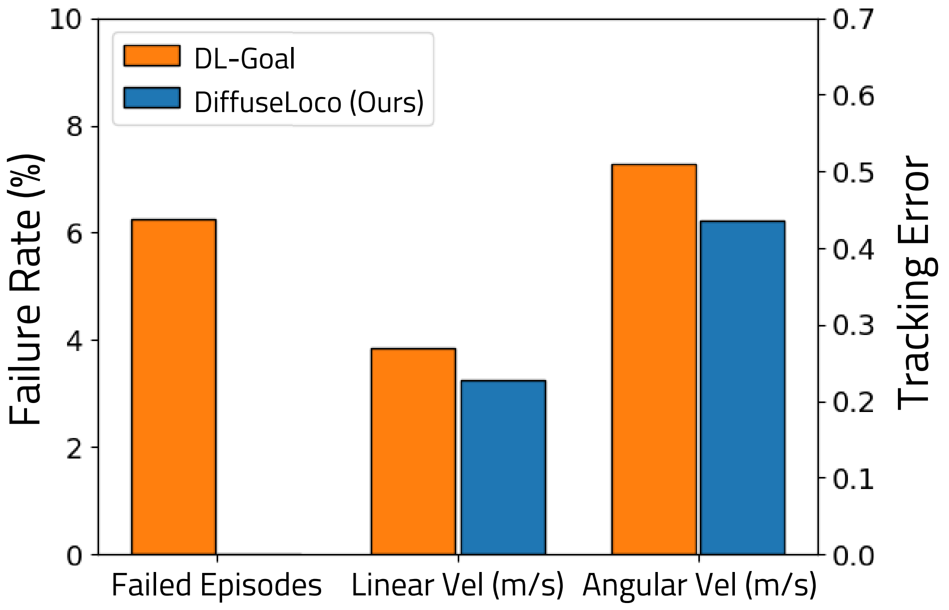

Figure 4.2: Comparison of failure rates and tracking errors between **DL w/o Goal** and **DiffuseLoco** (ours) in simulation. The left y-axis is the metric for Failed Episodes. The right y-axis indicates the tracking error for linear velocity and angular velocity.

– 100 Training + 10 Inference (**DDIM-100/10**)

– 10 Training + 5 Inference (**DDIM-10/5**)

Compared with our **DiffuseLoco**, **DDIM-100/10** has the same inference steps, and **DDIM-10/5** has the same training steps.

4.3.2 Single-step output versus RHC

RHC is an appropriate method to improve smoothness in diffusion-based policies. To minimize compounding variables, we also introduce a variant of DiffuseLoco without RHC (**DL w/o RHC**) to confirm the effects of RHC on learning legged locomotion control. As shown in Table 4.2, we see that without RHC, diffusion-based policy also fails most of the hard tasks and shows significant jittering behaviors that have high-frequent low-magnitude oscillation.

This result shows that single-step token-prediction models, such as the widely-used GPT models, might not be a suitable choice in imitating learning for legged locomotion control, whereas models designed for action sequence prediction, such as diffusion models, are a better fit.

4.3.3 Use of Goal-conditioning

We now assess the importance of using goal-conditioning in DiffuseLoco. We hypothesize that goal-conditioning primarily enhances tracking performance and stability in dynamic system control. In order to thoroughly assess tracking performance in a higher resolution, especially considering the noisy estimation of base velocity on real robots, we shift our focus to simulation environments with extensive randomization of dynamics. As demonstrated in Fig. 4.2, **DiffuseLoco** achieves a reduction of 15.4% decrease in linear velocity tracking error and 14.5% in that of angular velocity when compared to **DL w/o Goal** baseline. More importantly, over 64 trials with the same commands, **DL w/o Goal** falls over four times, or 6.25% of all trials, while **DiffuseLoco** does not experience any failure. This trend is also witnessed in real-world testing, where **DL w/o Goal** fails one trial in 0.7 m/s forward test.

These results underscore the importance of adding goal-conditioning with different attention weights for dynamic system control as shown in diagram 2.3. It is because the robot's I/O and goal possess distinct features, one governed by the law of physics while the other being an arbitrary objective, and thus could not be fused into one embedding space. An interesting research direction would be further investigating input elements with different features critical for control.

4.3.4 DDPM versus DDIM

As discussed earlier, popular diffusion-based frameworks often leverage DDIM to reduce sample iterations for inference acceleration. DDIM tends to trade off worse output quality with fewer, often 10 times less sampling iterations [56].

While this is a practical approach on most tasks that allow some variances such as image generation and manipulation tasks, we find that this acceleration method leads to reduced performance on control on dynamic systems, such as the quadrpeds used in this work. As shown in Table 4.2, both **DDIM-100/10** and **DDIM-10/5** show worse stability and velocity tracking error. The variant with 10 training and 5 inference steps fails one trial in the 0.7 m/s forward task, and the variant with 100 training steps and 10 inference steps shows more limpy behavior and fails one trial in the 0.5 m/s forward task as well. The tracking error compared to **DiffuseLoco** also increases by 50.69% and 42.04%, respectively.

This is likely because the noisier control signals produced by DDIM pipeline negatively affect controlling the floating-based dynamic systems (like legged robots) that are inherently less stable. In these cases, we need diffusion results of better quality without compromising for faster computing. The inference acceleration can be realized by other methods introduced in this work. Therefore, we do not recommend leveraging DDIM as an acceleration method for real-time locomotion control.

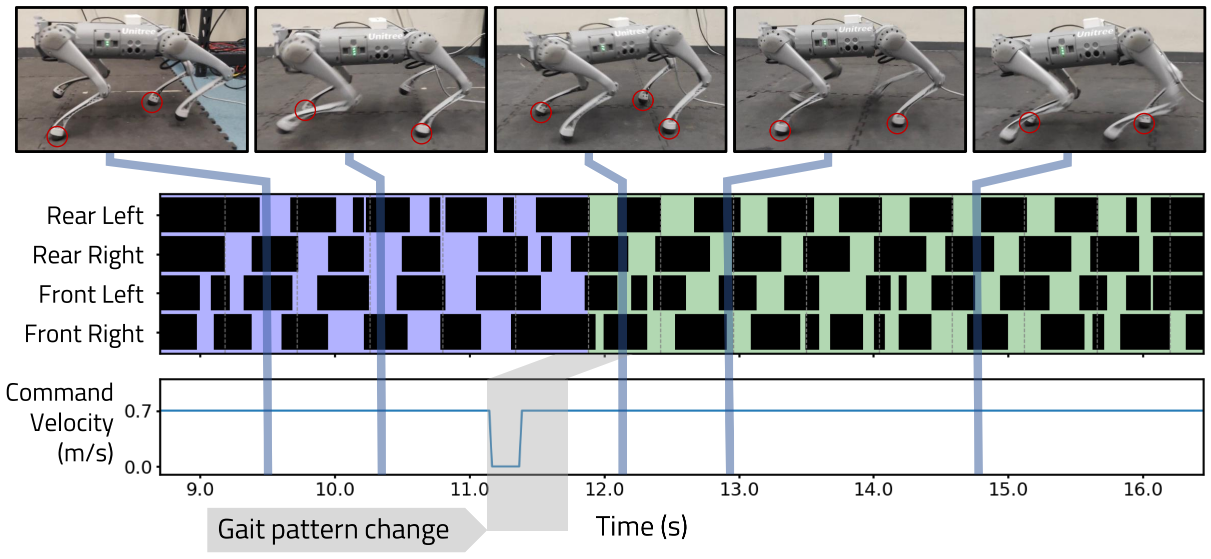

Figure 4.3: Foot contact map indicating stable walking and skill switching with DiffuseLoco policy and velocity commands. The red circle denotes the legs that are in contact with the ground. The change in command velocity indicates a sudden stop command and resuming to the original command. The robot walks with trotting skill during the first few steps, marked with a purple background, and then switches to pacing skill, marked with a green background. Upon receiving a stop command, the robot reduces speed by grounding three feet, coincidentally placing it in a posture such that it seamlessly continues to walk with the pacing skill upon command resumption.

4.3.5 Dataset Effects

Last, we investigate how the characteristics of the dataset might affect the robustness and performance of **DiffuseLoco** in the real world. Previous works show that adding more diversity by inserting noise helps mitigate compounding error [33]. Similarly, we show that increasing the variety of dynamics parameters of the environments to collect the dataset also helps improve robustness. As shown in Table 4.2, we train DiffuseLoco on a dataset of identical size and source policies with no dynamics randomization enabled in simulation. We see that both the robustness and stability could decrease by 44.26% using **DL w/o Rand**, indicating a large reduction in robustness. On the harder task of 0.7 m/s forward especially, the robot fell over in 3 out of the 5 trials using **DL w/o Rand**. This ablation shows us a future direction to investigate altering dataset, either adding more diversity or adding more fault-recovering behavior to further improve DiffuseLoco robustness.

4.3.6 Summary of Results

In the conclusion of this ablation study, we summarize several key design choices that enable better control performance using the diffusion models. In DiffuseLoco,

(1) use RHC instead of generating single action step to have smoother behavior in legged locomotion controls,

(2) separate the embeddings for robot's I/O and goal to better model the different features in these two spaces by transformer, resulting in better stability and tracking performance,

(3) use DDPM to preserve the best action sequence generation quality, while using other methods introduced in this work to accelerate the inference, and

(4) diversify the distribution of the collected offline dataset, such as incorporating dynamics randomization in simulation, to improve the robustness of IL-based policy.

4.4 Further Evaluation in Experiments

After benchmarking and ablation study, we perform further experiments to test the robustness and versatility of DiffuseLoco policy. First, we demonstrate that DiffuseLoco can efficiently learn various skills and switch among them smoothly on hardware. Later, we showcase the robustness of DiffuseLoco by walking on simple terrains and different grounds. Last, we extend DiffuseLoco to a bipedal robot, Cassie, and demonstrate walking policy in high-fidelity simulation.

4.4.1 Skill Switching

To showcase the advantages of using a DDPM, we highlight the ability of DiffuseLoco to maintain and switch skills under the *same* goal (commanded velocity) on a quadrupedal robot. In this test, the robot is commanded at a forward velocity of 0.7 m/s, followed by a sudden switch to 0 m/s, then quickly resume to the original forward velocity.

As shown in Fig. 4.3, the robot starts the test with a trotting skill. Then, the sudden brake brings disturbances to the contact sequence as the robot tries to reduce its forward velocity. When resuming, the robot leverages the new contact sequence and switches smoothly to pacing skill under the same command of 0.7 m/s. Importantly, the robot is able to leverage both pacing and trotting skills to walk *stably* before and after the switching. This shows that DiffuseLoco is able to learn the multi-modality existing in the dataset well and transfer successfully to the real world.

This differentiates us from AMP-based multi-skill policy in prior works that require different command velocity [18] or explicit skill conditioning [61, 66] for switching between skills.

4.4.2 Robustness Test

To evaluate the robustness of the DiffuseLoco policy, we perform four experiments on different grounds with various friction coefficients and small variations of the ground landscape.

Besides the padded floor in previous experiments, the DiffuseLoco policy is able to walk normally on different ground conditions: vinyl composite floor with low friction and low restitution, and artificial turf with low friction and high restitution, as shown in Fig. 4.4.3 and Fig. 4.4.3, respectively. We attribute this to the fact that our proposed dataset, which is composed of data with randomized dynamics, helps the policy to generalize and stay robust to varying ground conditions in the real world.

DiffuseLoco can also overcome a small variation in the terrain landscape. As Fig. 4.4.3 and Fig. 4.4.3 illustrate, the robot is able to step up and down a small step and ascend a low-angle ramp. Note that these variations in the terrain landscape is out of distribution of the offline dataset as we only simulate the robot walking on the flat ground when collecting the dataset. We attribute such robustness of DiffuseLoco to these unseen dynamics to the fact that DiffuseLoco, being a generative model, does not only learn specific trajectories from the dataset but also approaches the overall distribution of feasible actions, such that it is able to adapt to small changes in the terrain's dynamics.

4.4.3 Extension to Bipedal Locomotion Control

To demonstrate the versatility of our policy, we also train and deploy a DiffuseLoco policy(13.6M parameters) for the person-sized bipedal robot Cassie. A bipedal robot such as Cassie is a much more challenging control problem due to the high dimensional and highly nonlinear system. We collect the training dataset with single-skill walking policy trained with RL [37] and evaluate on high-fidelity simulation constructed in Matlab Simulink environment. Note that the data collected to train the DiffuseLoco policy is from MuJoCo, another simulator used to train the RL source policy and is less accurate in simulating rigid contacts.

As shown in Fig. 1.1, during the sim-to-sim transfer, the robot is able to maintain balance and follow the commanded velocity. This shows that our proposed approach is extensible to controlling the more challenging bipedal robots.

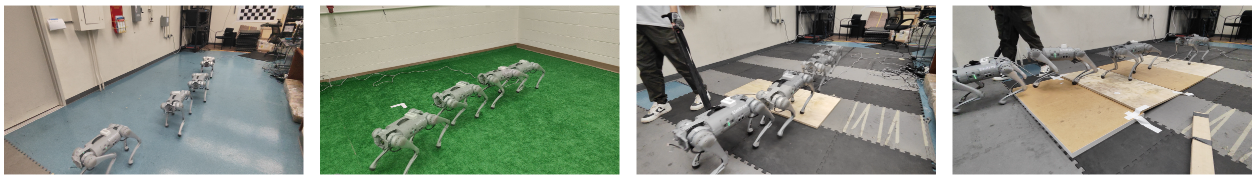

Figure 4.4: Depiction of our DiffuseLoco policy overcoming different terrains that are out of distributions of the collected offline dataset (as a flat-ground is used in during data collecting in simulation). (a): vinyl composite floor (b): grass terrain (c): foam padded floor with a wooden board as obstacle (d): foam padded floor with an inclined wooden board as a small variation in the terrain height.

Chapter 5

Conclusion and Discussion

5.1 Conclusion

This thesis presents DiffuseLoco, a novel offline imitation learning framework to learn dynamic legged locomotion *control* from multi-modal datasets and can transfer to real-world robots in real-time. DiffuseLoco leverages diffusion models to capture the multi-modality existing in the offline dataset, a challenge difficult to solve by commonly-used RL frameworks. To achieve real-time control based on diffusion, we propose new components and design choices, which enable DiffuseLoco to run on an edge-computing device onboard the robot. Extensive ablation studies validated the advantages brought by the proposed design choices. The proposed DiffuseLoco policy also outperforms baselines in extensive real-world benchmarks and demonstrates multi-modality in executing and switching skills smoothly when conditioned only with the same input commands. Moreover, DiffuseLoco policy shows robustness against different ground conditions and small variations of the ground landscape. In addition, we show further extension of applying DiffuseLoco to bipedal robots walking in high-fidelity simulation, but we note the failure to transfer the DiffuseLoco to a bipedal robot in the real world. Further increasing the robustness of the DiffuseLoco policy for more challenging control scenarios is an important future direction of DiffuseLoco. We deem this work as a new possibility for a more scalable and versatile framework for learning-based locomotion control in general.

5.2 Discussion

In this section, we provide a brief discussion on the advantages and disadvantages of the proposed DiffuseLoco, for a general takeaway for readers. Furthermore, we point out several exciting future routes based on DiffuseLoco for a more general control solution for legged locomotion or other complex nonlinear systems.

5.2.1 Advantages of DiffuseLoco

5.2.1.1 More Reliable Sim-to-Real Transfer for Multi-Skill Locomotion

We show that DiffuseLoco produces more reliable sim-to-real transfer performance among multi-skill AMP policies. While AMP policies suffer from a large sim-to-real gap resulting from the common problem of mode-collapsing in generator-discriminator style methods, DiffuseLoco avoids this problem and exhibits a stable, coherent, and effective multi-skill control. Furthermore, we show that DiffuseLoco demonstrates smooth skill-switching and stable skill execution under the same commands, highlighting the multi-skill performance achieved via the diffusion-based policy utilized in DiffuseLoco.

5.2.1.2 Less Overfitting from Offline Dataset

Shown in Sec. 4.2.3, we find that compared to Behavior Cloning baselines, our DiffuseLoco trained with DDPM loss is less likely to suffer from overfitting, leading to smoother actions and better stability and velocity tracking performance in real-world testing. This is further validated in Sec. 4.4.2 where DiffuseLoco is able to walk on grounds with different frictions and restitutions, and exhibits robustness against simple terrains not present in the dataset.

5.2.1.3 Real-time Control from Edge Computing

Lastly, we highlight the key advantage of DiffuseLoco: performing real-time feedback control utilizing 6.8M transformer-based diffusion models from a portable edge computing device. Through the technique of delayed observation and our acceleration pipeline, we are able to run DiffuseLoco policy at a frequency of 116.5 Hz, allowing plenty of space for scaling to larger models and datasets.

5.2.2 Limitations

A key limitation of this work is its lack of robustness compared to single-skill RL policies. In previous benchmarks, our DiffuseLoco policy is compared with multi-skill RL policies and demonstrates better robustness. However, against skill-specific RL policies, the robustness of our multi-skill DiffuseLoco policy is insufficient. For instance, a quadrupedal robot controlled by the DiffuseLoco policy struggles with recovery from large external perturbations, while task skill-specific RL policies handle effectively.

5.3 Future Work

5.3.0.1 Large-scale offline dataset for Locomotion

DiffuseLoco demonstrates the possibility of learning completely from offline datasets and deploying zero-shot transfer to real-world robots. We have witnessed the success of scaling up data in computer vision and natural language processing, and in the future, we wonder if it is also possible to create a large-scale locomotion dataset that is collected from various sources, including learning-based and model-based low-level controllers, for diverse tasks on various robots, and if it leads to the emergence of general intelligence for legged robot locomotion.

5.3.0.2 A generalist policy for locomotion control tasks

DiffuseLoco shows a promising direction of learning multiple, diverse locomotion skills in a single policy without the need for skill labeling through the multi-modal capacity of diffusion models. In future work, we hope to scale up the data diversity and the model size, like never been done before, to create a generalist policy as a foundation model for locomotion controls.

Bibliography

[1] Anurag Ajay et al. "Is conditional generative modeling all you need for decision-making?" In: *arXiv preprint arXiv:2211.15657* (2022).

[2] P. Arena et al. "MPC-based control strategy of a neuro-inspired quadruped robot". In: *2021 International Joint Conference on Neural Networks (IJCNN)* (2021), pp. 1–8. DOI: `10.1109/IJCNN52387.2021.9533394`.

[3] Kevin Black et al. "Zero-shot robotic manipulation with pretrained image-editing diffusion models". In: *arXiv preprint arXiv:2310.10639* (2023).

[4] Anthony Brohan et al. *RT-1: Robotics Transformer for Real-World Control at Scale.* 2023. arXiv: `2212.06817 [cs.RO]`.

[5] Anthony Brohan et al. *RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control.* 2023. arXiv: `2307.15818 [cs.RO]`.

[6] Tom Brown et al. "Language models are few-shot learners". In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901.

[7] Ju-Seung Byun and Andrew Perrault. "Training Transition Policies via Distribution Matching for Complex Tasks". In: *arXiv preprint arXiv:2110.04357* (2021).

[8] Huayu Chen et al. *Score Regularized Policy Optimization through Diffusion Behavior.* 2023. arXiv: `2310.07297 [cs.LG]`.

[9] Jianyu Chen, Bodi Yuan, and Masayoshi Tomizuka. "Deep imitation learning for autonomous driving in generic urban scenarios with enhanced safety". In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).* IEEE. 2019, pp. 2884–2890.

[10] Lili Chen, Shikhar Bahl, and Deepak Pathak. "Playfusion: Skill acquisition via diffusion from language-annotated play". In: *Conference on Robot Learning.* PMLR. 2023, pp. 2012–2029.

[11] Lili Chen et al. *Decision Transformer: Reinforcement Learning via Sequence Modeling.* 2021. arXiv: `2106.01345 [cs.LG]`.

[12] Wenshuo Chen et al. "Decomposed Human Motion Prior for Video Pose Estimation via Adversarial Training". In: *ArXiv* abs/2305.18743 (2023). DOI: `10.48550/arXiv.2305.18743`.

[13] Yiwen Chen et al. "FIRL: Fast Imitation and Policy Reuse Learning". In: *ArXiv* abs/2203.00251 (2022). DOI: `10.48550/arXiv.2203.00251`.

[14] Xuxin Cheng, Ashish Kumar, and Deepak Pathak. "Legs as Manipulator: Pushing Quadrupedal Agility Beyond Locomotion". In: *arXiv preprint arXiv:2303.11330* (2023).

[15] Cheng Chi et al. "Diffusion policy: Visuomotor policy learning via action diffusion." In: *arXiv preprint arXiv:2303.04137* (2023).

[16] Felipe Codevilla et al. "End-to-end driving via conditional imitation learning". In: *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2018, pp. 4693–4700.

[17] Ricard Durall et al. "Combating mode collapse in gan training: An empirical analysis using hessian eigenvalues". In: *arXiv preprint arXiv:2012.09673* (2020).

[18] Alejandro Escontrela et al. "Adversarial motion priors make good substitutes for complex reward functions". In: *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2022, pp. 25–32.

[19] Zipeng Fu, Xuxin Cheng, and Deepak Pathak. "Deep whole-body control: learning a unified policy for manipulation and locomotion". In: *Conference on Robot Learning*. PMLR. 2023, pp. 138–149.

[20] Zipeng Fu, Tony Z Zhao, and Chelsea Finn. "Mobile aloha: Learning bimanual mobile manipulation with low-cost whole-body teleoperation". In: *arXiv preprint arXiv:2401.02117* (2024).

[21] Huy Ha, Pete Florence, and Shuran Song. "Scaling up and distilling down: Language-guided robot skill acquisition". In: *Conference on Robot Learning*. PMLR. 2023, pp. 3766–3777.

[22] Jonathan Ho, Ajay Jain, and Pieter Abbeel. *Denoising Diffusion Probabilistic Models*. 2020. arXiv: `2006.11239 [cs.LG]`.

[23] Siyuan Huang et al. "Diffusion-based generation, optimization, and planning in 3d scenes". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023, pp. 16750–16761.

[24] Xiaoyu Huang et al. "Creating a dynamic quadrupedal robotic goalkeeper with reinforcement learning". In: *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2023, pp. 2715–2722.

[25] Xiaoyu Huang et al. "Skill transformer: A monolithic policy for mobile manipulation". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2023, pp. 10852–10862.

[26] Jemin Hwangbo et al. "Learning agile and dynamic motor skills for legged robots". In: *Science Robotics* 4 (2019). DOI: `10.1126/scirobotics.aau5872`.

[27] Michael Janner et al. *Planning with Diffusion for Flexible Behavior Synthesis*. 2022. arXiv: `2205.09991 [cs.LG]`.

[28] Ajay Jain Jonathan Ho and Pieter Abbeel. "Denoising diffusion probabilistic models." In: *arXiv preprint arXiv:2006.11239* (2020).

[29] Ivan Kapelyukh, Vitalis Vosylius, and Edward Johns. "DALL-E-Bot: Introducing Web-Scale Diffusion Models to Robotics". In: *IEEE Robotics and Automation Letters* 8.7 (July 2023), pp. 3956–3963. ISSN: 2377-3774. DOI: `10.1109/lra.2023.3272516`. URL: `http://dx.doi.org/10.1109/LRA.2023.3272516`.

[30] J. Kaplan et al. "Scaling Laws for Neural Language Models". In: *ArXiv* abs/2001.08361 (2020).

[31] Ilya Kostrikov, Ashvin Nair, and Sergey Levine. "Offline reinforcement learning with implicit q-learning". In: *arXiv preprint arXiv:2110.06169* (2021).

[32] Aviral Kumar et al. "Conservative Q-Learning for Offline Reinforcement Learning". In: *ArXiv* abs/2006.04779 (2020).

[33] Michael Laskey et al. "Dart: Noise injection for robust imitation learning". In: *Conference on robot learning*. PMLR. 2017, pp. 143–156.

[34] Sergey Levine et al. "Offline reinforcement learning: Tutorial, review, and perspectives on open problems". In: *arXiv preprint arXiv:2005.01643* (2020).

[35] Xiang Li et al. *Crossway Diffusion: Improving Diffusion-based Visuomotor Policy via Self-supervised Learning*. 2024. arXiv: `2307.01849 [cs.RO]`.

[36] Zhongyu Li et al. "Reinforcement learning for robust parameterized locomotion control of bipedal robots". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 2811–2817.

[37] Zhongyu Li et al. *Reinforcement Learning for Versatile, Dynamic, and Robust Bipedal Locomotion Control*. 2024. arXiv: `2401.16889 [cs.RO]`.

[38] Kanglin Liu et al. "Spectral regularization for combating mode collapse in gans". In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 6382–6390.

[39] Gabriel B Margolis et al. "Rapid locomotion via reinforcement learning". In: *arXiv preprint arXiv:2205.02824* (2022).

[40] Utkarsh A. Mishra et al. *Generative Skill Chaining: Long-Horizon Skill Planning with Diffusion Models*. 2023. arXiv: `2401.03360 [cs.RO]`.

[41] Vaishnavh Nagarajan and J Zico Kolter. "Gradient descent GAN optimization is locally stable". In: *Advances in neural information processing systems* 30 (2017).

[42] Ashvin Nair et al. "Overcoming exploration in reinforcement learning with demonstrations". In: *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2018, pp. 6292–6299.

[43] Mitsuhiko Nakamoto et al. "Cal-QL: Calibrated Offline RL Pre-Training for Efficient Online Fine-Tuning". In: *arXiv preprint arXiv:2303.05479* (2023).

[44] Felipe Nuti, Tim Franzmeyer, and João F Henriques. "Extracting Reward Functions from Diffusion Models". In: *arXiv preprint arXiv:2306.01804* (2023).

[45] L. Pacheco and N. Luo. "Testing PID and MPC Performance for Mobile Robot Local Path-Following". In: *International Journal of Advanced Robotic Systems* 12 (2015). DOI: `10.5772/61312`.

[46] Yunpeng Pan et al. "Agile autonomous driving using end-to-end deep imitation learning". In: *arXiv preprint arXiv:1709.07174* (2017).

[47] Tim Pearce et al. *Imitating Human Behaviour with Diffusion Models.* 2023. arXiv: `2301.10677 [cs.AI]`.

[48] X. B. Peng et al. "Learning Agile Robotic Locomotion Skills by Imitating Animals". In: *ArXiv* abs/2004.00784 (2020). DOI: `10.15607/rss.2020.xvi.064`.

[49] Xue Bin Peng et al. "Amp: Adversarial motion priors for stylized physics-based character control". In: *ACM Transactions on Graphics (ToG)* 40.4 (2021), pp. 1–20.

[50] Ilija Radosavovic et al. "Learning Humanoid Locomotion with Transformers". In: *arXiv preprint arXiv:2303.03381* (2023).

[51] Moritz Reuss et al. *Goal-Conditioned Imitation Learning using Score-based Diffusion Policies.* 2023. arXiv: `2304.02532 [cs.LG]`.

[52] Robin Rombach et al. *High-Resolution Image Synthesis with Latent Diffusion Models.* 2022. arXiv: `2112.10752 [cs.CV]`.

[53] Stephane Ross, Geoffrey J. Gordon, and J. Andrew Bagnell. *A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning.* 2011. arXiv: `1011.0686 [cs.LG]`.

[54] Dohoon Ryu and Jong Chul Ye. *Pyramidal Denoising Diffusion Probabilistic Models.* 2022. arXiv: `2208.01864 [cs.CV]`.

[55] Laura Smith et al. "Learning and adapting agile locomotion skills by transferring experience". In: *arXiv preprint arXiv:2304.09834* (2023).

[56] Jiaming Song, Chenlin Meng, and Stefano Ermon. *Denoising Diffusion Implicit Models.* 2022. arXiv: `2010.02502 [cs.LG]`.

[57] Chen Sun et al. "Revisiting Unreasonable Effectiveness of Data in Deep Learning Era". In: *2017 IEEE International Conference on Computer Vision (ICCV)* (2017), pp. 843–852. DOI: `10.1109/ICCV.2017.97`.

[58] Chen Tessler, Guy Tennenholtz, and Shie Mannor. "Distributional policy optimization: An alternative approach for continuous control". In: *Advances in Neural Information Processing Systems* 32 (2019).

[59] *Open Neural Network Exchange.* `https://onnx.ai/`. Accessed: 2024-01-29. 2024.

[60] Francecso Vezzi et al. "Two-Stage Learning of Highly Dynamic Motions with Rigid and Articulated Soft Quadrupeds". In: *arXiv preprint arXiv:2309.09682* (2023).

[61] Eric Vollenweider et al. "Advanced skills through multiple adversarial motion priors in reinforcement learning". In: *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2023, pp. 5120–5126.

[62] Bingzheng Wang et al. *DiffAIL: Diffusion Adversarial Imitation Learning*. 2023. arXiv: `2312.06348 [cs.LG]`.

[63] Hsiang-Chun Wang, Shang-Fu Chen, and Shao-Hua Sun. "Diffusion Model-Augmented Behavioral Cloning". In: *arXiv preprint arXiv:2302.13335* (2023).

[64] Zhendong Wang, Jonathan J Hunt, and Mingyuan Zhou. *Diffusion Policies as an Expressive Policy Class for Offline Reinforcement Learning*. 2023. arXiv: `2208.06193 [cs.LG]`.

[65] Eric R Westervelt et al. *Feedback control of dynamic bipedal robot locomotion*. CRC press, 2018.

[66] Jinze Wu, Yufei Xue, and Chenkun Qi. "Learning multiple gaits within latent space for quadruped robots". In: *arXiv preprint arXiv:2308.03014* (2023).

[67] Zhou Xian et al. "Chaineddiffuser: Unifying trajectory diffusion and keypose prediction for robotic manipulation". In: *Conference on Robot Learning*. PMLR. 2023, pp. 2323–2339.

[68] Wei Xiao et al. "SafeDiffuser: Safe Planning with Diffusion Probabilistic Models". In: *arXiv preprint arXiv:2306.00148* (2023).

[69] Haoran Xu et al. "A policy-guided imitation approach for offline reinforcement learning". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 4085–4098.

[70] Long Yang et al. *Policy Representation via Diffusion Probability Model for Reinforcement Learning*. 2023. arXiv: `2305.13122 [cs.LG]`.

[71] Ruihan Yang et al. "Multi-task reinforcement learning with soft modularization". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 4767–4777.

[72] Takuma Yoneda et al. "To the Noise and Back: Diffusion for Shared Autonomy". In: *arXiv preprint arXiv:2302.12244* (2023).

[73] Tianhe Yu et al. "Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning". In: *Conference on robot learning*. PMLR. 2020, pp. 1094–1100.