EPFL

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

RSL
Robotic Systems Lab

Master Thesis

Continuous Skill Learning for the ANYmal Robot

Spring Term 2024

Supervised by:
Chenhao Li
Dr. Nikita Rudin
Skandar Moalla
Prof. Dr. Marco Hutter
Prof. Dr. Caglar Gulcehre

Author:
Chengkun Li

Contents

Preface iii

Abstract v

Symbols vii

1 Introduction 1
- 1.1 Background . . . 1
- 1.2 Problem Formulation . . . 2
- 1.3 Related Work and Proposed Framework . . . 3
- 1.4 Contributions . . . 5
- 1.5 Thesis Organization . . . 5

2 Skill Latent Model with Sequence Models 7
- 2.1 Overview . . . 7
- 2.2 Design and Implementation . . . 7
  - 2.2.1 LSTM and GRU as RNN . . . 8
  - 2.2.2 State Space Models as RNN . . . 9
- 2.3 Training Skill Latent Model . . . 12
- 2.4 Experiments . . . 14
  - 2.4.1 Comparison between Sequence Models. . . . 15
  - 2.4.2 Effects of Skill Latent Space Structure on Generative Quality . . . 16
- 2.5 Summary . . . 18

3 Partial State-Conditioned Flow Adapter 19
- 3.1 Overview . . . 19
- 3.2 Theory . . . 20
- 3.3 Design and Implementation . . . 21
- 3.4 Training with Skill Latent Model . . . 22
- 3.5 Experiments . . . 24
  - 3.5.1 Effects of Skill Latent Space Structure . . . 24
- 3.6 Summary . . . 26

4 Policy Learning of Continuous Skill Learning Framework 27
- 4.1 Policy Learning Formulation . . . 27
- 4.2 Components . . . 28
- 4.3 Policy Learning Framework . . . 28
  - 4.3.1 Skill Latent Policy . . . 28
  - 4.3.2 Learned Action Blending . . . 29
  - 4.3.3 Refinement Policy . . . 29
- 4.4 Configuration . . . 30
- 4.5 Testing on Multiple Terrains . . . 31
- 4.6 Analyses and Studies . . . 36

4.7 Summary 40

5 Conclusion **41**
5.1 Summary 41
5.2 Future Work 41

Bibliography **46**

A Policy Configuration **47**

B Terrain Description **49**

Preface

The completion of this thesis marks the culmination of my (i.e. Chengkun Li) master's journey in the Robotics program at EPFL. The majority of the work was carried out at the Robotic Systems Lab at ETH Zürich and the CLAIRE Lab at EPFL, under the official co-supervision of Professor Dr. Marco Hutter and Professor Dr. Caglar Gulcehre. At ETH Zürich, I had the privilege of working closely with my supervisors Chenhao Li and Dr. Nikita Rudin, and at EPFL, I had the pleasure to work with Skander Moalla.

I am especially grateful to my daily supervisors, Chenhao Li, Dr. Nikita Rudin, and Skander Moalla, for their constant guidance, insightful advice, and patience throughout every stage of this thesis. Your feedback and mentorship have been invaluable in shaping both my research and personal growth as a scholar. Thank you for always being available, for pushing me to reach my full potential, and for the countless brainstorming sessions and experiment analyses. Working with you has been one of the most rewarding aspects of this journey.

I would also like to sincerely thank my hosting supervisor at ETH Zürich, Professor Dr. Marco Hutter, for providing me with the opportunity and great resources to pursue this research. My deepest thanks also go to my co-hosting supervisor at EPFL, Professor Dr. Caglar Gulcehre, for your invaluable insights, patience, and the warm welcome whenever I returned to Lausanne—it truly made me feel like I was back home.

A special mention goes to my colleagues, professors, and friends at EPFL, ETH Zürich, and Google Zürich, who filled my academic journey with collaboration, fulfillment, humor, shared challenges, and companionship. I could never have imagined the richness of this experience before coming to Switzerland for my master's studies.

To my family, no words can fully express my gratitude. Thank you for your love, patience, and unwavering encouragement throughout this journey. To my parents, thank you for teaching me resilience and determination. And to my brother, thank you for always being there to offer insights and feedback, both in my academic life and beyond.

Thank you all!

Yours sincerely,
Chengkun
Sept 2024, Zürich

Abstract

This work presents a novel framework for continuous skill learning tailored for the ANYmal quadruped robot. The proposed approach leverages the principles of skill decomposition, recombination, and refinement to enable the robot to efficiently acquire, adapt, and generalize new motor skills by building on past experiences. To achieve this, various sequence models, including Long Short-Term Memory (LSTM), Gated Recurrent Units (GRU), and state-space models, are explored for encoding skills within a continuous latent space. Additionally, a Partial State-Conditioned Flow Adapter is introduced to structure the latent space, enabling more efficient exploration and skill selection during policy learning.

The core contribution of this work is the development of a two-tier policy architecture, which integrates a skill latent policy that uses previously learned skills with a refinement policy designed to fine-tune and adapt these skills to new tasks and environments. Empirical validation is conducted across a diverse set of terrains, both seen and unseen during training, demonstrating the robot's ability to refine its skills dynamically and efficiently.

This research has the potential to advance the field of continuous learning in robotics, providing new insights into lifelong learning, skill representation, and the potential applications of structured latent spaces for enhancing robotic adaptability and performance in complex environments.

Symbols

Symbols

v_b	Base linear velocity
ω_b	Base angular velocity
g_b	Projected gravity
p_t^c	Position command
ψ_t^c	Heading command
t_G^*	Time left to target
$q_t, \dot{q}_t$	Joint positions and velocities
a_{t-1}	Decoded joint command of previous step
h	Heightmap scan ($2m \times 1m$)
z_{t-1}	Skill latent of previous step

Acronyms and Abbreviations

NVP	Non-Volume Preserving
SSM	State Space Model
RNN	Recurrent Neural Network
LSTM	Long Short-Term Memory
GRU	Gated Recurrent Units
MLP	Multilayer perceptron
FA	Flow Adapter
SLM	Skill Latent Model

Chapter 1

Introduction

Humans and animals possess a remarkable ability to continually acquire, refine, and transfer knowledge and skills throughout their lives. This ability to learn and adapt enables them to respond effectively to changing environments and new challenges, increasing their chances of survival [1, 2]. For instance, humans constantly learn new skills for various roles or adapt to new tools and technologies, often leveraging previously acquired knowledge to accelerate this process. Quick learners, in particular, excel at integrating new information by drawing on their existing knowledge base, demonstrating a capacity for rapid adaptation and flexible thinking.

In the realm of robotics and artificial intelligence, the most effective agents should similarly be capable of ongoing learning and adaptation. An ideal robotic or artificial agent would not merely learn fixed tasks; instead, it would continuously improve and adapt, retaining what it has learned while also discerning when and how to apply this knowledge effectively. Such an agent would be able to learn dynamically, much like a quick learner in humans, constantly updating its knowledge base and adapting to new circumstances.

This thesis aims to develop a framework for continuous skill learning from an egocentric perspective, focusing on the ANYmal quadruped robot. The objective is to leverage recent advances in representation learning, sequence modeling, and lifelong learning to create a system that mimics the adaptability and continuous learning capabilities observed in biological organisms. Specifically, the proposed approach seeks to enable the robot to acquire new skills by building on past experiences and refining its skills based on new understanding of tasks, thereby integrating both new and existing knowledge effectively.

1.1 Background

Continuous Learning. Building continuous learning machines has been a long-standing goal [1, 3], with key features including the ability to transfer and reuse knowledge to adapt rapidly to novel environments without extensive retraining. A continuous learning system should support both forward and backward transfer of information. Forward transfer refers to the ability of previously learned knowledge to facilitate the acquisition of new knowledge, enabling the application of mastered skills to new tasks. In contrast, backward transfer occurs when learning new tasks enhances performance on previously learned, related tasks. This ability to refine and leverage existing knowledge is crucial for robust and efficient continuous learning. Additionally, continuous learning systems must address catastrophic forgetting, where neural networks overwrite prior knowledge when trained on new data, and exploit task similarities to accelerate learning and improve generalization. In this study, we aim to develop a continuous skill learning framework for the ANYmal robot, striving to achieve the beneficial properties of a continuous learning system.

Legged Locomotion. Legged locomotion has experienced significant advancements in the past decade, particularly in the domain of learning-based methods. This progress has been driven by the convergence of several key factors: the rise of deep learning, the rapid development of high-fidelity and efficient physics simulators, and the availability of robust, affordable hardware. On-policy deep reinforcement learning algorithms, such as TRPO and PPO, have been especially effective in generating robust and agile locomotion behaviors for quadrupeds, often surpassing traditional model-based approaches. Researchers are integrating exteroceptive data, such as LiDAR, camera, directly into control policies, enabling robots to navigate complex, unstructured terrains. Moreover, techniques like domain randomization, system identification, and domain adaptation are helping bridge the sim-to-real gap, facilitating the successful transfer of learned policies to physical robots. However, there are still open challenges, including efficient skill discovery, navigating increasingly difficult environments, safe and reliable deployment of these advanced robots, etc.

ANYmal Robot. ANYmal is a series of highly capable quadruped [4], in this work we build our method around the ANYmal D which weighs around $55\ kg$ and has 12 series elastic actuators each producing up to $85\ Nm$.

Sequence Models. Recent advancements in sequence modeling have largely been driven by two major classes of models: recurrent neural networks (RNNs) and attention-based models. Among RNNs, Long Short-Term Memory (LSTM) networks [5] and Gated Recurrent Units (GRUs) [6] have been particularly influential due to their ability to mitigate the vanishing and exploding gradient problem [7, 8] and maintain memory over relatively longer sequences. By incorporating mechanisms such as gating and memory cells, LSTMs and GRUs enable the retention and manipulation of information over extended sequences, effectively capturing long-range dependencies that are crucial for tasks like language modeling, speech recognition, and time series prediction.

In the context of sequence models, a critical trade-off exists between efficiency and effectiveness, defined by the model's ability to compress its state. Efficient models are characterized by a compact state, while effective models maintain a state that encapsulates all essential information from the context. Attention-based models, most notably the Transformer [9] architecture, have revolutionized sequence modeling by fundamentally altering how information is processed and represented. Transformers utilize self-attention mechanisms to directly model dependencies between all elements of a sequence, regardless of their distance, thereby dispensing with the sequential processing nature inherent to RNNs. The ability of attention mechanisms to capture global context has led to their widespread adoption in a variety of applications, especially in natural language processing (NLP) [10, 11], computer vision [12, 13], and more recently, in the modeling of discrete sequences in bioinformatics like proteins [14]. However, while Transformers excel in capturing global dependencies in discrete and structured data, they are less suited for continuous signals [15, 16] and relies on customized tokenizers for specific use cases [17, 18] and suffer from quadratic complexity in sequence length.

1.2 Problem Formulation

In this work, we formulate continuous skill learning as an ongoing process of adaptation and refinement. The agent initially applies its previously learned skills to new tasks or environments and then refines these skills based on updated insights. This process mirrors the learning behaviors observed in egocentric agents, such as humans and other living organisms. By leveraging past knowledge, the agent enhances its adaptability and efficiency in acquiring new skills.

The goal of this thesis is to develop a framework for continuous, egocentric skill learning. The framework is structured around the following key principles:

1. **Skill Decomposition**: Each skill can be broken down into smaller components, or "skill snippets" with certain horizon length which represent the fundamental building blocks of more complex skills.
2. **Skill Snippet Recombination**: New and more complex skills are formed by recombining these basic skill snippets, allowing the agent to adapt flexibly and creatively to different tasks.
3. **Leveraging Past Experiences**: Utilizing previously acquired skills enables the agent to quickly adapt to new environments. This starting point accelerates the learning process, even when existing knowledge is insufficient to fully solve new tasks.
4. **Continuous Refinement**: After applying prior experience to a new task, the agent continues to improve its skills, leading to greater rewards and adaptability. This ongoing refinement is crucial for enhancing long-term performance.

We will show in the following chapters how we incorporate these principles into the design of continuous skill learning framework for ANYmal robot.

1.3 Related Work and Proposed Framework

The framework for continuous skill learning in this work is illustrated in Figure 1.1. This process is guided by the principle of learning from past experiences and refining based on those experiences and refining of learning with a episodic memory [19]. The accumulated knowledge from past interactions is encoded in a skill latent model, which is then used as a foundation for ongoing learning and refinement to conquer new tasks.

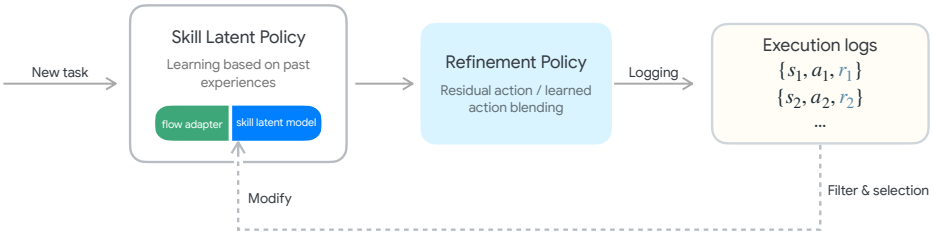


Figure 1.1: The continuous skill learning framework consists of three components: a skill latent policy that learns from past experiences, a refinement policy that adapts the skill latent policy's output to the current environment, and a periodic log memory for storing execution logs that may be useful for future learning.

When presented with a task, the agent within the continuous skill learning pipeline initiates by deriving a skill latent policy informed by the skill latent model (as detailed in Section 4.3.1). Subsequently, a refinement policy is employed to enhance the performance of the skill latent policy. This refinement is achieved either through the modulation of parameters within the skill latent policy or by the direct application of residual actions to its output. Once the task is completed, the execution logs are recorded. A portion of these logs is then used to fine-tune either the flow adapter or the skill latent model. The flow adapter, being parameter-efficient, is lighter, whereas the skill latent model is more computationally intensive.

Our work is closely related to skill learning, quadrupedal locomotion learning, motion representation learning, and lifelong learning in general. Specifically, we take inspiration from the work in the skill prior and skill embedding community and build a framework around some of the core ideas from skill learning, and delved further into the representation

learning aspect of skill embedding (skill latent in our work), and further investigate the limitation and potential lifelong learning aspect of the framework.

Skill embedding and Skill prior. Skill-Prior Reinforcement Learning (SPiRL) [20] is a pioneering work that proposes a deep latent model to jointly learn an embedding space of skills and a skill prior from offline agent experience. This approach significantly accelerates the learning of downstream tasks by leveraging previously learned skills. SPiRL learns a continuous skill embedding using a deep latent variable model, encoding a sequence of actions into a low-dimensional skill space. It further learns a skill prior, which is a conditional probability of a skill latent given some conditioning factors like environment, goals, or sensory information. This prior guides exploration and decision-making in downstream tasks, focusing on more promising areas of the skill space. Another relevant work, ReSkill [21], builds upon SPiRL by introducing a state-conditioned skill prior and a low-level residual policy. The state-conditioned prior improves exploration efficiency by sampling relevant skills based on the current state. The residual policy allows for fine-tuning skills to adapt to unseen task variations, enhancing robustness and flexibility. Hausman *et al.* [22] introduces an entropy-regularized policy gradient formulation for hierarchical policies and a data-efficient off-policy gradient algorithm. This approach learns a skill embedding space where skill clusters are encouraged to cover the space, promoting versatility and transferability.

Quadrupedal Locomotion Learning. Policies Modulating Trajectory Generators [23] introduce an approach that combines a policy with a parametric trajectory generator for quadrupedal gaits. This allows the policy to modulate trajectory parameters and directly output control commands, enabling adaptable speed and robust walking behaviors with minimal sensory input. Hierarchical reinforcement learning has been adopted in several works [24, 25, 26], where multiple expert policies are trained—often through imitation learning or carefully designed reward functions. A high-level policy is then trained to coordinate and switch between these expert policies. While this approach is effective and to some extent scalable, it relies on smooth and rapid transitions between policies to maintain responsive and fluid actions, and if additional policies are introduced or existing ones are modified, the high-level policy requires retraining to ensure cohesive integration of the updated policy set.

Motion representation learning. Motion representation learning plays a crucial role in enabling robots to acquire, refine, and generalize complex movements. A central challenge in this field is determining the most effective representation for diverse tasks, encompassing both locomotion and manipulation. Current approaches can be broadly categorized into physics-based and kinematics-based methods. *Physics-based motion latent spaces* often utilize adversarial learning or motion tracking to form latent representations of movements. Adversarial methods, as exemplified by ASE [27], learn to generate realistic motions by mapping noise to behavior through a discriminator network. However, these methods can be limited in their ability to represent diverse skills. Explicit motion tracking approaches, such as ControlVAE [28] and NPMP [29], train models to reconstruct motion capture data, typically using imitation learning in conjunction with reinforcement learning. These methods can struggle with generalization to novel tasks or environments. A recent work [30] proposes a universal humanoid motion representation by distilling knowledge from an imitator trained on a large-scale motion dataset, demonstrating the potential for learning general-purpose motion representations. On the other hand, *Kinematics-based motion representations* learn representations directly from data without relying on physics simulation. Alternatively, approaches like PAE [31] and FLD [32] compress motions into periodic latent spaces, leveraging the inherent periodicity of many movements. These

kinematic representations offer computational efficiency but can struggle with representing complex interactions with the environment.

1.4 Contributions

This contributions of this work is mainly divided into four folds:

- A framework for continuous, egocentric skill learning: This framework, tailored for the ANYmal quadruped robot, integrates concepts ok skill decomposition, recombination, and refinement, enabling the robot to adapt and learn new skills efficiently by leveraging past experiences.
- Exploration and evaluation of three types of sequence models for skill encoding: LSTMs, GRUs, and state-space models like S4D, for encoding skill "muscle memory" within a continuous latent space. Implementation of a Partial State-Conditioned Flow Adapter that leverages Normalizing flow to transform the skill latent space into a more structured and policy-friendly latent space. This adapter aids in efficient exploration and context-aware skill selection during policy learning.
- Development and evaluation of a combined skill latent and refinement policy: The thesis presents a two-level policy architecture that combines a skill latent policy, leveraging previously learned skills, with a lightweight refinement policy for fine-tuning and adapting those skills to new tasks.
- Empirical validation of the framework on various terrains: The effectiveness of the proposed framework is demonstrated through its ability to learn and refine locomotion skills across diverse terrains, including both seen and unseen environments.

1.5 Thesis Organization

The main content of this work is structured into following chapters, each addressing a key component of the research on continuous skill learning for the ANYmal robot.

In Chapter 1 we introduce the motivation and scope of this research, and the core problems this thesis addresses, specifically in terms of developing a framework for skill learning that operates efficiently in latent space.

In Chapter 2 we delve into the design of skill latent model, which encodes skill snippets into a continuous latent space. The chapter describes how this model leverages sequence models such as LSTM, GRU, and state-space models to capture the temporal dynamics of motor skills. A comparison between these models is provided, detailing their respective strengths and limitations, empirical results are given on the hyperparameters for training.

Chapter 3 introduces the flow adapter, a lightweight and parameter-efficient mechanism designed to fine-tune the skill latent space. The flow adapter guides the policy toward relevant skill latents, making exploration more focused and computationally efficient. By modifying the latent space based on the partial state of the robot, it reduces the need for extensive retraining of the skill latent model. The theoretical basis for the flow adapter is discussed, followed by empirical results on the hyperparameters for training.

Chapter 4 integrates the components described in earlier chapters - the skill latent model and flow adapter, into a unified skill latent policy. Together with a lightweight refinement policy, this framework is used to train policies that operate both in the skill latent space and in low-level action spaces, facilitating efficient learning and adaptation to new tasks. The chapter provides a detailed explanation of how policies are trained, alongside experimental results that evaluate the system's performance across various terrains. Comparisons with baseline methods are presented, demonstrating the advantages of the proposed framework in terms of both learning speed and adaptability.

Chapter 2

Skill Latent Model with Sequence Models

Our method models skill learning in two stages: first, by searching within the original muscle memory space, and second, by recombining, refining, and building upon these learned muscle memories. This chapter focuses on how we model the latent space for encoding these acquired muscle memories for future learning.

2.1 Overview

The skill latent model is designed to encode "muscle memory" for skills within a continuous latent space. When learning a new task, the model navigates this latent space to determine if parts of existing skills can be applied to the new problem. Thus, the skill latent model must effectively represent skills in a way that facilitates both exploration and learning by the policy network. This requires careful design of the model architecture and the selection of suitable loss functions to ensure that the skill representations are robust and adaptable.

2.2 Design and Implementation

The skill latent model adopts an encoder-decoder architecture, where the input signal is first compressed into a lower-dimensional latent space and then reconstructed to approximate the original input data as closely as possible.

Notice

From this point onward, we will refer to the agent's partial state as s, which in our design comprises joint positions, joint velocities, and the current gravity direction. This notation will be frequently used in the following designs of skill latent model (this chapter) and flow adapter (in Chapter 3).

All skill action data in the dataset is segmented into time series with a defined horizon length, H. Each segment, or "skill snippet," is represented as $x = \{a_1, a_2, \ldots, a_H\}$, where each step a_i consists of a joint target position command obtained from policy rollouts. Given a skill snippet of length H, the recurrent encoder processes the input sequence according to the specific type of RNN used and generates a skill latent variable $\mathbf{z}$ through variational autoencoding. This latent variable $\mathbf{z}$ encapsulates the entire skill snippet in a

compressed form. Subsequently, the skill latent $\mathbf{z}$ serves as the input to the recurrent decoder, which reconstructs the sequence by decoding it step-by-step.

2.2.1 LSTM and GRU as RNN

The vanilla version of the skill latent model relies on two well-established variants of recurrent neural networks (RNNs): Long Short-Term Memory (LSTM) networks and Gated Recurrent Units (GRUs). Both LSTM and GRU models have been extensively studied and are widely recognized for their ability to effectively maintain and update a hidden state over time, which is crucial for modeling skill data with a horizon length.

Long Short-Term Memory. An LSTM cell maintains two states: the hidden state h_t and the cell state c_t, and cell state acts as a memory and preserves information over long context, while the hidden state provides a representation of the output for the current time step. Each LSTM cell computes

$$\begin{aligned} i_t &= \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi}) \\ f_t &= \sigma(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf}) \\ g_t &= \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg}) \\ o_t &= \sigma(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho}) \\ c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\ h_t &= o_t \odot \tanh(c_t) \end{aligned} \tag{2.1}$$

where i_t, f_t, g_t, o_t are the input, forget, cell, and output gates, respectively. h_t is the hidden state at time t, c_t is the cell state at time t, x_t is the input at time t, h_{t-1} is the hidden state of the layer at time $t-1$ or the initial hidden state at time 0. σ is the sigmoid function, and $\odot$ is the Hadamard element-wise product.

Gated Recurrent Unit. GRU is a simpler variant of the LSTM designed to achieve similar performance while reducing computational complexity, each GRU cell computes

$$\begin{aligned} r_t &= \sigma(W_{ir}x_t + b_{ir} + W_{hr}h_{(t-1)} + b_{hr}) \\ z_t &= \sigma(W_{iz}x_t + b_{iz} + W_{hz}h_{(t-1)} + b_{hz}) \\ n_t &= \tanh(W_{in}x_t + b_{in} + r_t \odot (W_{hn}h_{(t-1)} + b_{hn})) \\ h_t &= (1 - z_t) \odot n_t + z_t \odot h_{(t-1)} \end{aligned} \tag{2.2}$$

where r_t, z_t, n_t are the reset, update, and new gates, respectively. h_t is the hidden state at time t, x_t is the input at time t, $h_{(t-1)}$ is the hidden state of the layer at time $t-1$ or the initial hidden state at time 0. σ is the sigmoid function, and $\odot$ is the Hadamard element-wise product. Thus, the state is maintained only by the hidden state, reducing the number of parameters compared to the LSTM.

Building the Model. For the sake of clarity, we first show how to construct the skill latent encoder-decoder utilizing LSTM or GRU units as the foundational RNN building blocks, as depicted in Figure 2.1. The encoder processes the action at each time step, a_t, and propagates the hidden state to the next step. Depending on the chosen RNN architecture, either the cell state c_t of the LSTM or the hidden state h_t of the GRU is maintained throughout this process. The final hidden state is then used as a compact representation of the entire sequence. For the variational autoencoding component, we employ two Multi-Layer Perceptrons (MLPs) to map this sequence representation into two vectors, μ (mean) and σ (standard deviation), capturing the probabilistic nature of the latent space. This approach allows the model to learn a compressed, informative representation of the input sequences.

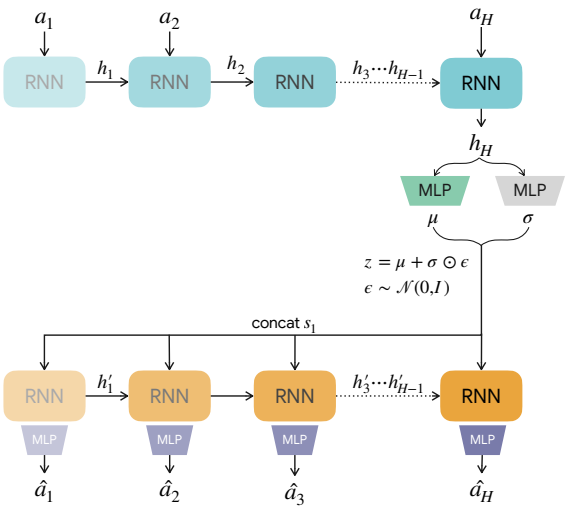


Figure 2.1: Structure of skill latent encoder-decoder. From left to right means time unrolling direction

Additionally, we keep the option of incorporating a β-VAE framework, which introduces a tunable hyperparameter β to balance the trade-off between reconstruction fidelity and latent space regularization. By adjusting β, we can potentially enhance the disentanglement of the learned latent representations, providing flexibility to tailor the model's learning dynamics as needed. In literature that utilize β hyperparamters, usually the value is set to a relatively small value to ensure reconstruction quality while still making sure the latent space is still well regularized.

During decoding, the latent variable z is concatenated with the robot's initial state s_1, serving as a conditioning input to the RNN decoder at each time step. This conditioning with s_1 ensures that the decoder maintains awareness of the initial context, which is crucial for reconstructing future actions based on the initial state. For subsequent time steps, the RNN decoder takes the previous hidden state and continues to use the concatenation of z and s_1 as input. This setup results in the output being the predicted action $\hat{a}_t$ at each time step.

2.2.2 State Space Models as RNN

State Space Models (SSMs) [15] have emerged as powerful sequence modeling frameworks, particularly effective for tasks requiring the handling of long-range dependencies [33]. These models are designed to parameterize transformations between time series signals, $u(t) \mapsto y(t)$, drawing on concepts from classical control theory and signal processing. In machine learning, SSMs have been successfully adapted to function as RNNs, leveraging their natural ability to model temporal dynamics through an evolving internal state representation.

The foundational framework of SSMs as sequence models is often grounded in the principles of Linear Time-Invariant (LTI) systems. An LTI system is defined by a set of linear differential equations that govern the evolution of a latent state $x(t)$ over time. The general mathematical representation of an SSM in the context of an LTI system is given by:

$$\begin{aligned} x'(t) &= \mathbf{A}x(t) + \mathbf{B}u(t), \\ y(t) &= \mathbf{C}x(t) + \mathbf{D}u(t), \end{aligned} \tag{2.3}$$

where the input signal $u(t)$, a one-dimensional function, is mapped to an N-dimensional

latent state $x(t)$ and subsequently projected back to produce the output signal $y(t)$ with state transition matrix $\mathbf{A} \in \mathbb{R}^{N\times N}$, input matrix $\mathbf{B} \in \mathbb{R}^{N\times 1}$, output matrix $\mathbf{C} \in \mathbb{R}^{1\times N}$, and feedthrough matrix $\mathbf{D} \in \mathbb{R}$.

In recent developments within machine learning, SSMs are parameterized by matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$, while the matrix $\mathbf{D}$ is frequently omitted [15, 34]. The term $\mathbf{D}u(t)$ can be interpreted as a skip connection that directly propagates the input signal to the output, enhancing model flexibility and reducing redundant parameterization. It has been shown that naively instantiate matrix $\mathbf{A}$ does not perform well [35] and the optimization of the state matrix $\mathbf{A}$ in SSMs usually leverages the HiPPO (High-order Polynomial Projection Operators) framework [36]

HiPPO defines a class of matrices specifically crafted to efficiently capture and maintain information about continuous functions. By initializing $\mathbf{A}$ with HiPPO matrices, SSMs demonstrate strong performance in retaining long-range dependencies over extended sequences.

Applying to the discrete input signal $(u_0, u_1, u_2, \ldots)$ instead of continuous input function like $u(t)$, the state space model in Equation (2.3) should be discretized with a step size parameter Δ which serves as the resolution of the input and the discrete inputs u_k as an implicit sampling of the continous signal $u(t)$ at time $k\Delta$, i.e. $u_k = u(k\Delta)$. The discretization of SSMs can be achieved through various methods, such as the forward/backward Euler, Zero-Order Hold (ZOH), or bilinear (Tustin) discretization [37], which approximate the continuous dynamics using finite differences. After applying discretization (here we use bilinear), the SSM formulation becomes

$$\begin{aligned} x_k &= \bar{\mathbf{A}} x_{k-1} + \bar{\mathbf{B}} u_k, \\ y_k &= \bar{\mathbf{C}} x_k, \\ \bar{\mathbf{A}} &= \left(\mathbf{I} - \frac{\Delta}{2}\cdot\mathbf{A}\right)^{-1}\left(\mathbf{I} + \frac{\Delta}{2}\cdot\mathbf{A}\right), \\ \bar{\mathbf{B}} &= \left(\mathbf{I} - \frac{\Delta}{2}\cdot\mathbf{A}\right)^{-1}\Delta\mathbf{B}, \\ \bar{\mathbf{C}} &= \mathbf{C}. \end{aligned} \tag{2.4}$$

And the discretized formulation can now be viewed as a recurrence in x_k resembling a RNN. The recurrent formulation Equation (2.4) is not practical for training on modern hardware due to its sequential nature, and the convolutional form is often used to make SSM easy to train with modern hardware. By unrolling the discrete formulation Equation (2.4) and let the initial state be $x_{-1} = 0$, we can have

$$\begin{aligned} &x_0 = \overline{\mathbf{B}}u_0; \quad x_1 = \overline{\mathbf{AB}}u_0 + \overline{\mathbf{B}}u_1; \quad x_2 = \overline{\mathbf{A}}^2\overline{\mathbf{B}}u_0 + \overline{\mathbf{AB}}u_1 + \overline{\mathbf{B}}u_2; \quad \ldots \\ &y_0 = \overline{\mathbf{CB}}u_0; \quad y_1 = \overline{\mathbf{CAB}}u_0 + \overline{\mathbf{CB}}u_1; \quad y_2 = \overline{\mathbf{CA}}^2\overline{\mathbf{B}}u_0 + \overline{\mathbf{CAB}}u_1 + \overline{\mathbf{CB}}u_2; \quad \ldots \end{aligned} \tag{2.5}$$

thus can be formulated as a convolution operation

$$\begin{aligned} y_k &= \overline{CA}^k\overline{B}u_0 + \overline{CA}^{k-1}\overline{B}u_1 + \cdots + \overline{CAB}u_{k-1} + \overline{CB}u_k \\ y &= \overline{\mathbf{K}} * u \\ \overline{\mathbf{K}} &= \left(\overline{CB}, \overline{CAB}, \ldots, \overline{CA}^{L-1}\overline{B}\right). \end{aligned} \tag{2.6}$$

where the SSM convolutional kernel $\overline{\mathbf{K}} \in \mathbb{R}^L$ can be computed efficiently using the fast Fourier transforms (FFTs).

Structured State Space Sequence Models (S4). While State Space Models (SSMs) with HiPPO matrices are highly effective at capturing long-range dependencies, they often incur

significant computational costs, particularly due to repeated matrix multiplications involving $\overline{\mathbf{A}}$, as seen in Equation (2.6). To address these computational challenges, the Structured State Space Sequence Model (S4) [15] introduces a more structured and efficient parameterization. A key innovation in S4 is the use of the HiPPO-LegT (Legendre Transform) basis, which is designed to approximate continuous functions by projecting them onto a sequence of Legendre polynomials. This transform is adapted to align better with the discrete settings of sequence modeling. The HiPPO-LegT state matrix $\mathbf{A}$ is defined as follows:

$$\mathbf{A}_{nk} = -\begin{cases} (2n+1)^{1/2}(2k+1)^{1/2} & \text{if } n > k \\ n+1 & \text{if } n = k \\ 0 & \text{if } n < k. \end{cases} \tag{2.7}$$

Furthermore, S4 incorporates a decomposition of the state matrix $\mathbf{A}$ into a Normal Plus Low-Rank (NPLR) form:

$$\mathbf{A}_{\text{S4}} = \mathbf{A}_{\text{N}} - \mathbf{P}_{\text{S4}}\mathbf{Q}_{\text{S4}}^{T}, \tag{2.8}$$

where $\mathbf{A}$N represents the normal (diagonalizable) component, and $\mathbf{P}$S4 and $\mathbf{Q}_{\text{S4}}$ are the low-rank components. This parameterization improves computational efficiency by reducing the complexity of matrix operations. The full paramterization of this NPLR representation is

$$\mathbf{A} = \mathbf{V}\Lambda\mathbf{V}^{*} - \mathbf{P}\mathbf{Q}^{\top} = \mathbf{V}\left(\Lambda - (\mathbf{V}^{*}\mathbf{P})(\mathbf{V}^{*}\mathbf{Q})^{*}\right)\mathbf{V}^{*}, \tag{2.9}$$

where $\mathbf{V} \in \mathbb{C}^{N\times N}$ is the unitary matrix that diagonalizes $\mathbf{A}$, Λ is the diagonal matrix of eigenvalues, $\mathbf{P}, \mathbf{Q} \in \mathbb{R}^{N\times r}$ are the low-rank factors, with r representing the rank (for HiPPO-LegT, $r = 1$). Thus, the total number of parameters in the S4 layer consists of the diagonal matrix Λ and the vectors $\mathbf{P}, \mathbf{Q}, \mathbf{B}, \mathbf{C} \in \mathbb{C}^{N\times 1}$. This results in a total parameter count of $5N$, making S4 computationally efficient while still capable of modeling long-range dependencies effectively. S4 operates with H independent 1-dimensional SSMs, each processed by an independent S4 mapping. The outputs from these mappings are then combined using a position-wise linear layer, followed by a non-linearity layer to produce the final result.

S4D (Diagonal version of S4). A further exploration based on S4 state space models led to the development of S4D [38], a diagonal version of S4. S4D achieves comparable performance to S4 by restricting the state matrix to be fully diagonal (removing the low rank term in Equation (2.8), significantly simplifying the implementation and computation. The diagonal initialization, such as in S4D-Inv given by:

$$\mathbf{A}_{n} = -\frac{1}{2} + i\frac{N}{\pi}\left(\frac{N}{2n+1} - 1\right), \tag{2.10}$$

where the real part controls the decay rate, and the imaginary part governs oscillatory behavior. This structured initialization ensures the system remains stable while effectively modeling long-range dependencies. Similar to S4, S4D operates with H independent 1-dimensional State Space Models (SSMs), where each SSM processes the input sequence independently. The outputs of these individual SSMs are then combined through a position-wise mixing layer, which plays a crucial role in aggregating the information from multiple SSMs. The mixing layer is a linear transformation applied to the outputs at each position of the sequence, allowing the model to integrate contributions from each SSM and produce a unified output.

Building the Skill Latent Model with S4D. We adopt the same encoder-decoder plus variational bottleneck structure as described in Section 2.2.1, with the core components replaced by the S4D model, as shown in Figure 2.2. In this architecture, the input is first transformed into a higher-dimensional space using a linear layer, then processed by the S4D

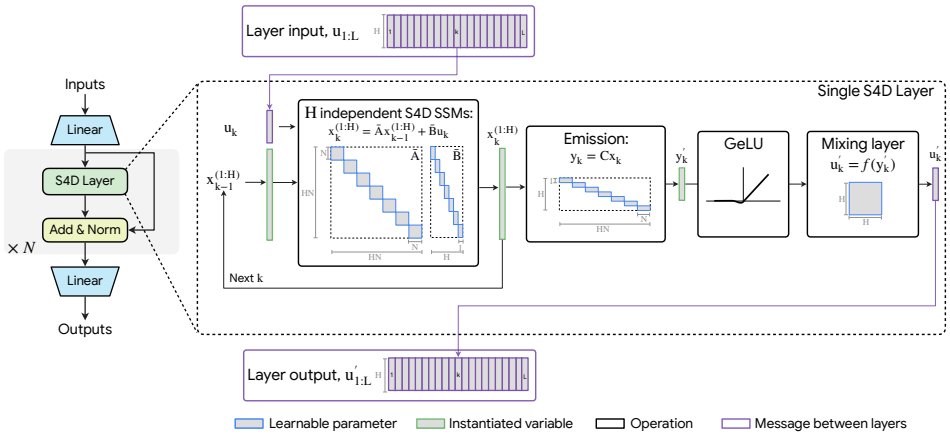


Figure 2.2: S4D Encoder structure. Inspired by the visualization in [16], the input of dimension H is processed by H parallel SSMs. For non-linearity, GeLU [39] is employed. Mixing layer is applied to mix outputs from these H SSMs.

layer. A residual connection is applied, followed by normalization and down-sampling. It is important to note that the input includes a dimension corresponding to sequence length. Since the aim is to produce a latent vector independent of the sequence length (i.e., encoding the entire sequence into a single latent vector, similar to extracting the final hidden state or memory state in previous models), the last step of the sequence output is used to represent the entire encoded sequence, which is then passed into the variational bottleneck.

After obtaining this final representation of the entire skill sequence, the latent vector is duplicated along the sequence length dimension and concatenated with the initial state s_1 (similar to Section 2.2.1. This concatenated representation, which now includes both the repeated latent vector and the state, serves as the input to the decoder. By this design, we ensure that the decoder has access to both the full skill sequence representation and the state information across all time steps, enhancing the model's ability to reconstruct the original sequence.

It's worth noting that by adopting this approach, we avoid materializing the intermediate hidden states within the S4D layer, which are represented as complex numbers and would be computationally expensive to store and manipulate in the convolutional setup. Instead, by using only the last step output to summarize the sequence, the computational complexity and memory demands are significantly reduced.

The complete encoder-decoder structure, incorporating the variational bottleneck, is presented in Figure 2.3.

> **Sidenote:** It's also worth mentioning that, while we assume the encoder and decoder use the same base model in this section, the interface between the encoder and decoder is designed in a unified manner, allowing them to be used with arbitrary combinations. This flexible interface supports different model architectures while maintaining compatibility. However, due to time constraints in this study, we default to using matching encoder and decoder models. Future work could explore a wider variety of combinations.

2.3 Training Skill Latent Model

The formulation of the training loss plays a crucial role in sequence learning, as it directly impacts the effectiveness of the policy learning that follows. As we employ a variational

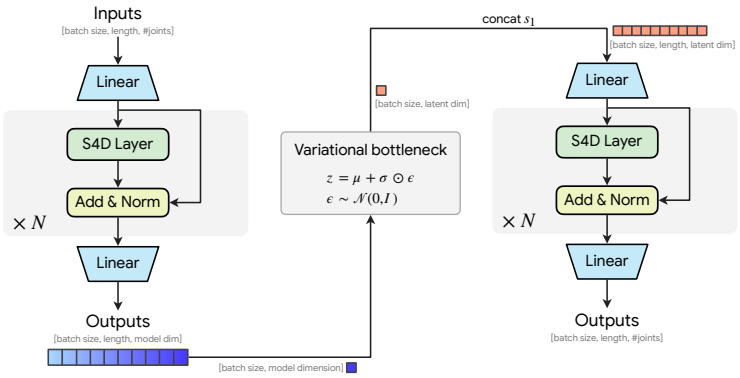


Figure 2.3: Skill Latent Encoder-Decoder Structure with S4D. The encoder and decoder share the same structure, where the encoder only takes the last step output before feeding it into the variational bottleneck. The decoder takes the entire output as input, with the latent vector duplicated along the sequence length dimension and concatenated with the state.

autoencoding bottleneck, our objective is to encode the latent representations in a continuous space. Simultaneously, to ensure that the latent space maintains sufficient separation between different skills, we incorporate a form of contrastive loss [40]. This approach ensures that distinct skills at different states are encoded separately, while similar skills at similar states are clustered together, which facilitates more efficient policy network training. The overall loss function is therefore a weighted sum of the variational autoencoding loss and the contrastive loss, expressed as:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{variational}} + \lambda \mathcal{L}_{\text{contrastive}} \tag{2.11}$$

Variational autoencoding loss. We adapt the standard loss function framework introduced in the Variational Autoencoder (VAE) [41] and β-VAE[42] to better suit our model by incorporating additional conditioning on the initial state. The modified loss function is expressed as:

$$\mathcal{L}\left(x^{(i)}, s_1^{(i)}, \theta, \phi\right) = \underbrace{-\mathbb{E}_z\left[\log p_\theta\left(x^{(i)} \mid z, s_1^{(i)}\right)\right]}_{\text{Reconstruction}} + \underbrace{\beta \times D_{KL}\left(q_\phi\left(z \mid x^{(i)}\right) \parallel p_\theta\left(z\right)\right)}_{\text{Latent Regularization}} \tag{2.12}$$

In this formulation, the reconstruction term is modified to condition the reconstruction of the input data $x^{(i)}$ not only on the latent variable z but also on the initial state $s_1^{(i)}$. This modification ensures that the decoder can leverage information from the first state to improve the fidelity of the reconstruction, particularly in tasks where the initial state carries significant contextual information. The latent space regularization term remains a Kullback-Leibler (KL) divergence between the approximate posterior $q_\phi(z \mid x^{(i)})$ and the prior $p_\theta(z)$, weighted by a factor β. This term was designed to encourage the learned latent distribution to stay close to the prior, promoting structured and interpretable latent spaces. By adjusting β, we can control the trade-off between reconstruction quality and latent space regularization, aligning with the principles of the β-VAE framework.

Contrastive loss. We build a state-based contrastive loss function that is designed to guide the learning of latent representations by leveraging the similarities between state inputs. The primary objective of the contrastive loss is to encourage high cosine similarity between latent vectors whose corresponding state inputs are similar, while pushing latent

vectors of dissimilar state inputs further apart. This approach ensures that the learned latent representations reflect the structure and relationships present in the state inputs.

First, we define the key variables: z is the batch of latent vectors with $z \in \mathbb{R}^{B\times D}$, where B is the batch size and D is the latent dimension, and s is the batch of state inputs, with $s \in \mathbb{R}^{B\times d\times H}$, where d is the state dimension and H is the skill horizon. Cosine similarity, denoted as $\text{sim}(a, b) = \frac{a\cdot b}{\|a\|\|b\|}$, is used to measure similarity between pairs of vectors. We also introduce parameters such as the temperature τ, the positive similarity threshold θ_{pos}, and the negative similarity threshold θ_{neg}. We first normalize latent vectors as $z'_i = \frac{z_i}{\|z_i\|}$ and state inputs as $s'_i = \frac{s_i}{\|s_i\|}$. The state inputs are then sampled to focus on a limited horizon: $s''_i = \text{reshape}(s'_{i,:,1:\min(5,H)}) \in \mathbb{R}^{d\times(\min(5,H))}$. After normalization, we compute the latent similarity matrix $S_z(i, j) = \text{sim}(z'_i, z'_j)$ and the state similarity matrix $S_s(i, j) = \text{sim}(s''_i, s''_j)$. Using these similarity matrices, we define two masks: the positive mask $P(i, j) = 1[S_s(i, j) > \theta_{pos}]$, which identifies pairs of states that are similar with a threshold, and the negative mask $N(i, j) = 1[S_s(i, j) < \theta_{neg}]$, which identifies pairs that are dissimilar. For each latent vector z_i, we compute the average positive similarity as

$$\text{Pos}_i = \begin{cases} \frac{\sum_{j=1}^{B} S_z(i,j)\cdot P(i,j)}{\sum_{j=1}^{B} P(i,j)} & \text{if } \sum_{j=1}^{B} P(i, j) > 0 \\ 0 & \text{if no positive pairs exist} \end{cases}$$

and the average negative similarity as

$$\text{Neg}_i = \begin{cases} \frac{\sum_{j=1}^{B} S_z(i,j)\cdot N(i,j)}{\sum_{j=1}^{B} N(i,j)} & \text{if } \sum_{j=1}^{B} N(i, j) > 0 \\ 0 & \text{if no negative pairs exist} \end{cases}$$

The logits for each sample are constructed as $L_i = \left[\frac{\text{Pos}_i}{\tau}, \frac{\text{Neg}_i}{\tau}\right] \in \mathbb{R}^2$, where $\text{Pos}_i \in \mathbb{R}$ is the average positive similarity, $\text{Neg}_i \in \mathbb{R}$ is the average negative similarity, and τ is the temperature parameter. The label $Y_i = 0$ is set to indicate that the model should maximize the positive similarity (the first element in the logits vector). Thus, for a batch of size B, the logits matrix L will have dimensions $L \in \mathbb{R}^{B\times 2}$, where each row corresponds to the positive and negative similarity scores for each sample. The final contrastive loss is computed as the cross-entropy between the logits and the target labels:

$$\mathcal{L}_{\text{contrastive}} = \frac{1}{B}\sum_{i=1}^{B} \text{CrossEntropy}(L_i, Y_i), \tag{2.13}$$

where $L_i \in \mathbb{R}^2$ is the logits vector for the i-th sample, and $Y_i \in \mathbb{R}$ is the corresponding target label.

Dataset generation. To train the skill latent model, we collect low-level policy rollout from [25] by deploying pre-trained low-level policies on specific terrains within a simulated legged robot environment. It's worth noting, because we are training the skill latent model with the motor target position directly (control signal), therefore the skill latent model training data can be collected from other policies or controling algorithms as well. We collect policy rollouts of 3500 agents per environment-policy pair, and each rollouts is made up of segments with length $H = 5$, and the total amount of training data is $5.166M$ clips. The combination of environment-policy pair are: (climb down, climb down), (gaps, gaps), (climb box, climb), (slopes, walk), (stairs, stairs), and (tables, crouch).

2.4 Experiments

In this section, we compare the difference in terms of performance between different sequence models, and the effects of hyperparameters.

Configuration. In the experiments, we fixed the latent dimension to $d = 10$ and selected a skill horizon of $H = 5$. We evaluated model performance by comparing reconstruction quality as the primary evaluation metric. The reconstruction quality is measured by a variant of Mean Squared Error (MSE) between the input x and reconstruction $\hat{x}$ defined as:

$$\text{Reconstruction MSE Per Step} = \frac{\sum_{i=1}^{B}\sum_{t=1}^{H}\|x_t^{(i)} - \hat{x}_t^{(i)}\|^2}{BH} \tag{2.14}$$

where $x_t^{(i)}$ is the actual action at time step t for sample i and $\hat{x}_t^{(i)}$ is the reconstructed action at time step t for sample i. In our case, both vectors are 12 dimensional.

2.4.1 Comparison between Sequence Models.

We compared the performance between LSTM, GRU, and S4D based skill latent models, as described in Sections 2.2.1 and 2.2.2. To ensure a fair comparison, all sequence models were designed with a similar number of trainable parameters by adjusting the hidden state dimensions of each model. The KL regularization parameter β was fixed at 1×10^{-6}. Additionally, we excluded the contrastive loss from the evaluation in this experiment. The decision to omit the contrastive loss was based on its sensitivity to dataset shuffling, which can introduce undesirable variability into the experimental setup as the primary goal of this section is to evaluate the intrinsic learning capacity of the sequence models for sequential skill action data.

All models were trained using a batch size of 1024, a learning rate of 0.002, and a cosine annealing learning rate scheduler, over 100 epochs. During training, we applied a dropout rate of 0.05 to the initial state information of the decoder, simulating potential noise or data loss that could arise during real-world deployment on robots, where state estimation might be noisy or incomplete. The hidden state dimensions were carefully tuned to ensure all models had a comparable number of trainable parameters, allowing for an unbiased evaluation of each model's performance as a candidate for the skill latent model.

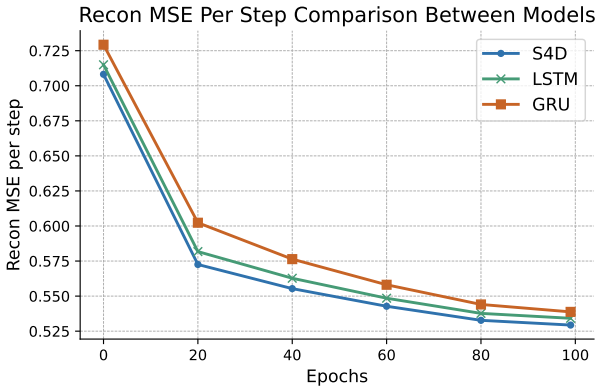


Figure 2.4: Comparison between sequence models: reconstruction MSE for the skill latent model using different building blocks (LSTM, GRU, and S4D).

The reconstruction mean squared error (MSE) per step, used as the primary evaluation metric, is presented in Figure 2.4. Additionally, we report the number of trainable parameters for each configuration in Figure 2.5. The results indicate that the S4D-based skill latent model consistently achieves comparable or better performance than the GRU and LSTM models while using fewer parameters, both in the overall model and in the decoder specifically. Given its performance and parameter efficiency, we select the S4D model as the default for subsequent experiments. Although, as mentioned in **??**, further exploration of other encoder-decoder combinations could yield additional insights.

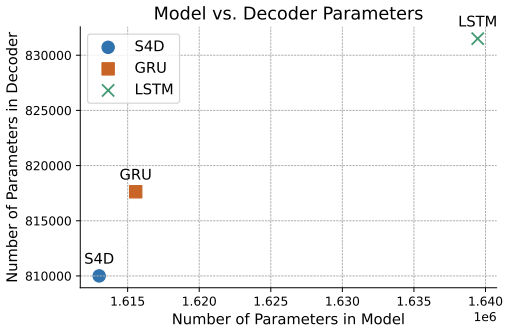


Figure 2.5: Comparison of parameter efficiency between sequence models for the skill latent model and decoder. The S4D model demonstrates superior parameter efficiency compared to GRU and LSTM, using fewer parameters while maintaining or slightly improving performance.

Ablation Study: Initial State Information To investigate the impact of including the initial state s_1 in the skill latent decoder, we conducted an ablation study. In this experiment, we compared the reconstruction performance of the skill latent model with and without the initial state s_1 as input to the decoder. We evaluated this design choice across three versions of the skill latent model (LSTM, GRU, and S4D). The results, shown in Figure 2.6, reveal a notable decrease in reconstruction quality when the initial state information is removed from the decoder's input.

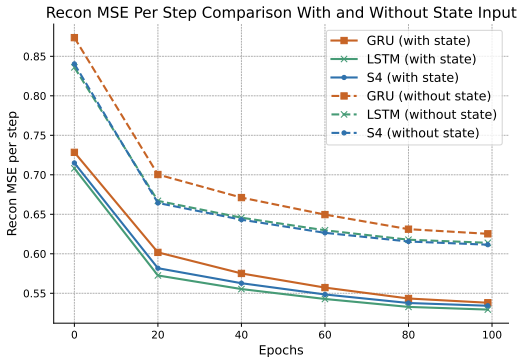


Figure 2.6: Impact of initial state information: Reconstruction MSE per step comparison with and without the initial state s_1 as input to the skill latent decoder.

2.4.2 Effects of Skill Latent Space Structure on Generative Quality

Sweep of the KL Regularization Term β. In this experiment, we examine the impact of varying the β value, a key parameter in the β-VAE framework, on the generative quality of the model. Specifically, we aim to understand how different levels of Kullback-Leibler (KL) regularization influence the organization of the skill latent space and, consequently, the model's reconstruction capability. The contrastive loss weight is set to zero throughout this experiment to eliminate variability caused by dataset shuffling, ensuring that we isolate the effects of β on the model's performance. The β-VAE framework introduces a KL divergence term that encourages the latent distribution to approximate a standard Gaussian prior. This regularization plays a crucial role in shaping the structure of the latent space. By im-

posing this constraint, the latent representations are encouraged to become more organized, which could potentially enhance downstream tasks like exploration and generalization in the latent space.

To evaluate the effect of β, we train the S4D-based skill latent model while sweeping β across the values $\{0, 1 \times 10^{-6}, 1 \times 10^{-5}, 1 \times 10^{-4}, 1 \times 10^{-3}\}$. The training setup remains fixed, except for the varying β values. Reconstruction mean squared error (MSE) per step is used as the primary metric to measure the quality of reconstruction.

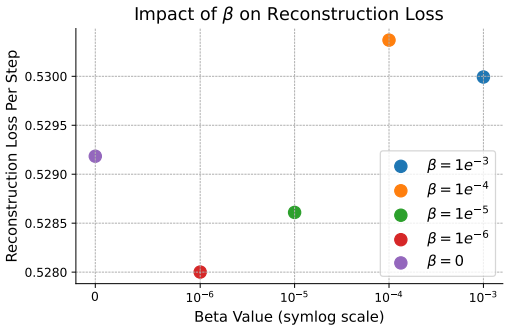


Figure 2.7: Comparison of reconstruction MSE per step across different values of β, which controls the strength of the KL regularization in the latent space. As β increases, the latent space is increasingly constrained towards a Gaussian prior, influencing the model's reconstruction quality. The results indicate that small values of β slightly improve or maintain performance compared to the unregularized model, while larger values do not provide significant benefits, β value larger than 1 tend to affect the reconstruction quality significantly.

Contrastive Weight (λ) Sweep. In a complementary experiment, we investigate the effect of varying the contrastive weight λ on reconstruction performance. The contrastive learning mechanism aims to enhance the model's ability to differentiate between similar and dissimilar representations by pushing apart negative examples and pulling together positive examples in the latent space. The goal of this experiment is to observe how increasing the contrastive weight influences the model's reconstruction capabilities and the contrastive loss itself.

We sweep λ over the range [0, 5], the contrastive loss measures how well the model preserves the similarities and differences between latent space representations measured by cosine similarity between states of horizon length H . As expected, Figure 2.8 shows a direct relationship between the contrastive weight and the contrastive loss: higher contrastive weights result in a lower contrastive loss, reflecting the increased emphasis on aligning similar representations. We also examine the impact of λ on reconstruction quality by plotting the reconstruction MSE per step against the contrastive weight, as shown in Figure 2.9. Interestingly, the R^2 value for the linear regression fit is close to zero, indicating no strong correlation between the contrastive weight and reconstruction quality. This suggests that the strength of the contrastive loss does not directly affect the model's ability to reconstruct the data, at least within the tested range of λ. Additionally, the data points on the running minimum curve differ only marginally from the performance when no contrastive loss is applied, with a difference of approximately 0.001 in the reconstruction MSE. This indicates that moderate levels of contrastive loss do not significantly alter the reconstruction ability of the model and the representations in the latent space are still able to preserve the necessary information for accurate reconstructions, even under moderate contrastive pressure.

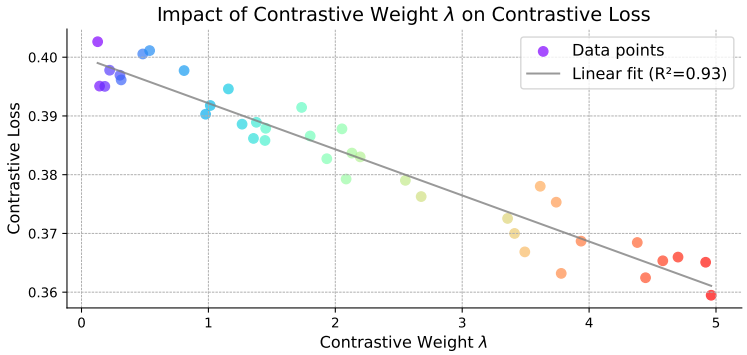


Figure 2.8: Contrastive loss as a function of the contrastive weight λ. As λ increases, the contrastive loss decreases, reflecting the increased focus on representation alignment in the latent space.

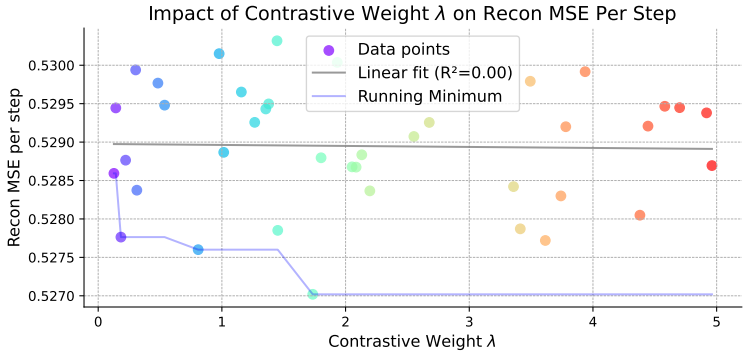


Figure 2.9: Reconstruction MSE per step as a function of contrastive weight λ. The results suggest that there is no strong correlation between the contrastive weight and reconstruction quality. Moderate contrastive regularization does not significantly affect the decoder's ability to reconstruct the data.

2.5 Summary

This chapter introduces the skill latent model, a core component of our method for modeling skill learning. This model aims to efficiently and effectively encode "muscle memory" of learned skills within a continuous latent space. By representing skills in this manner, the following up policy can facilitate the exploration and recombination of existing skills when faced with new tasks. We explored recurrent neural networks (LSTM, GRU) and state space models (S4D) as architectural candidates. Empirical evaluation demonstrated that the S4D-based model achieved superior performance and parameter efficiency compared to LSTM and GRU, while incorporating initial state information proved crucial for reconstruction quality. Further investigation into latent space regularization using β-VAE and a novel state-based contrastive loss revealed that moderate regularization improves performance, and the contrastive loss primarily impacts latent space organization without significantly affecting reconstruction. The S4D-based skill latent model provides a powerful and efficient foundation for subsequent skill learning stages.

Chapter 3

Partial State-Conditioned Flow Adapter

"Learning without thought is labor lost; thought without learning is perilous."

— *Confucius*

Learning with past experiences is not solely about having acquired knowledge stored in a skill memory bank—it is also about recognizing when and how to effectively apply that knowledge. A key challenge in skill learning lies in the ability to discern the right context in which to utilize stored skills, ensuring that they are explored and used appropriately and efficiently during policy learning.

3.1 Overview

In addition to the learned skill latent space, we introduce a Partial State-conditioned Flow Adapter (referred to as the "Flow Adapter" for simplicity), which is trained before and then utilized during policy learning. The system diagram for this flow adapter is shown in Figure 3.1. The flow adapter is based on a Real-valued Non-Volume Preserving (Real NVP) network [43], which consists of a series of coupling layers. Each coupling layer learns a parameterized bijective transformation, denoted as f, where the input and output maintain the same dimensionality, as illustrated in Figure 3.1. The key advantage of the Flow Adapter is its ability to preserve information about the skill latent space through invertible transformations while decluttering the structure of skill latent space based on the current state, thereby converting the latent space into a more policy-friendly action space.

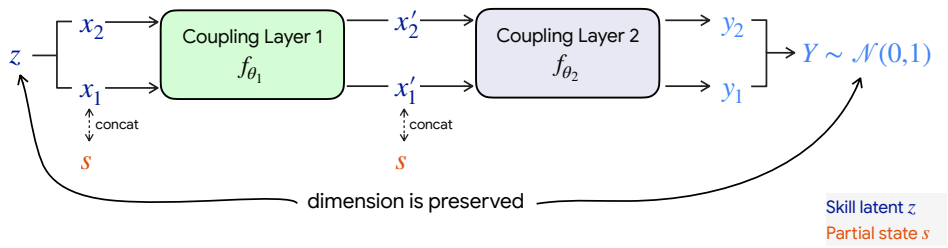


Figure 3.1: An example of a partial state-conditioned flow adapter with two coupling layers parameterized by θ_1 and θ_2, respectively. The input is first split into two vectors, x_1 and x_2, and then processed through two affine coupling layers.

The Flow Adapter plays a critical role in identifying **relevant skills** during policy learning by associating skill latent with their corresponding context (partial state). This allows the policy to better understand when and how to apply specific skills. During exploration in the skill latent space, the Flow Adapter guides the policy to focus on regions of the latent space that are most likely to contain useful skills for completing the task.

Additionally, the Flow Adapter is designed to be parameter-efficient and trained in self-supervised manner, requiring no additional data labeling. It operates independently of the skill latent model, allowing the system to function with or without the adapter—hence the name "adapter". This design flexibility ensures that the flow adapter can be integrated seamlessly without interfering with the training process of the skill latent model, as described in the previous chapter.

The primary function of the Flow Adapter is to learn a bijective mapping that supports both forward $(z, s) \rightarrow y$ and backward $z \leftarrow (y, s)$ transformations, where z represents the skill latent vector, s represents the partial state, and y is the output of the transformation. Typically, y is chosen to follow a simple distribution, such as a standard Gaussian. Once the invertible function f is learned, it can transform a simple distribution into a more complex one, enabling more efficient exploration during policy learning.

3.2 Theory

Normalizing flows [44, 45] represent a class of generative models that transform a simple base distribution (such as a Gaussian) into a more complex target distribution through a sequence of invertible and differentiable transformations.

The Real NVP model constructs its invertible transformation layers using *affine coupling layers*, where, in each bijection, part of the input vector is updated with a function that is straightforward to invert and depends on the remainder of the input vector. Specifically, for an input vector x of dimension D, the vector is split into two parts of dimensions d and $D - d$, respectively. The output y after an affine coupling layer $y = f(x, z)$ is defined as:

$$\begin{aligned} y_{1:d} &= x_{1:d} \\ y_{d+1:D} &= x_{d+1:D} \odot \exp(sc(x_{1:d})) + t(x_{1:d}) \end{aligned} \tag{3.1}$$

where $sc(\cdot)$ and $t(\cdot)$ are learnable functions for scaling and translation respectively, both from $\mathbb{R}^d \mapsto \mathbb{R}^{D-d}$, and $\odot$ is the element-wise product. The inverse can then be derived as:

$$x_{1:d} = y_{1:d} \tag{3.2}$$

$$x_{d+1:D} = (x_{d+1:D} - t(x_{1:d})) \odot \exp(-sc(x_{1:d})) \tag{3.3}$$

Notably, both the forward and inverse transformations involve only the evaluation of $sc(\cdot)$ and $t(\cdot)$, and not their inverses. This allows these functions to be implemented in a highly non-invertible and complex manner without hindering the computation of the inverse transformation.

Training the Real NVP model involves maximizing the log-likelihood of the observed data samples, denoted as x, under the probabilistic model $p_X(x)$. This process enables the model to learn a transformation function f that maps the data from the input space X to a latent space Y, such that $y = f(x)$. The transformation preserves the probability density of the data, and using the change of variables formula, the log-likelihood of $p_X(x)$ is expressed as:

$$p_X(x) = p_Y(f(x)) \left| \det \left(\frac{\partial f(x)}{\partial x^T} \right) \right| \tag{3.4}$$

$$\log(p_X(x)) = \log(p_Y(f(x))) + \log \left(\left| \det \left(\frac{\partial f(x)}{\partial x^T} \right) \right| \right), \tag{3.5}$$

where the second term, $\log\left(\left|\det\left(\frac{\partial f(x)}{\partial x^T}\right)\right|\right)$, accounts for the volume change induced by the transformation f, captured by the determinant of the Jacobian matrix of f. This determinant reflects how the transformation stretches or compresses different parts of the space.

In the case of Real NVP, the transformation function f consists of affine coupling layers, as described in Equation (3.1), where the Jacobian matrix takes the form of a lower triangular structure. The Jacobian for an affine coupling layer is:

$$\frac{\partial y}{\partial x^T} = \begin{bmatrix} \mathbf{I}_d & 0 \\ \frac{\partial y_{d+1:D}}{\partial x_{1:d}^T} & \text{diag}\left(\exp[sc(x_{1:d})]\right) \end{bmatrix}, \tag{3.6}$$

where $\mathbf{I}_d$ is the identity matrix corresponding to the first d dimensions of the input x, and $\text{diag}(\exp[sc(x_{1:d})])$ is a diagonal matrix whose entries are the exponentiated outputs of the learned scaling function $sc(x_{1:d})$. Because the Jacobian matrix is triangular, its determinant can be computed efficiently as the product of its diagonal elements, which simplifies to:

$$\det\left(\frac{\partial f(x)}{\partial x^T}\right) = \exp\left[\sum_j sc(x_{1:d})_j\right], \tag{3.7}$$

where $sc(x_{1:d})_j$ denotes the j-th element of the scaling function applied to the first d components of the input x. Consequently, the log-determinant is simply the sum of the scaling terms, leading to the final form of the log-likelihood:

$$\log(p_X(x)) = \log(p_Y(f(x))) + \sum_j sc(x_{1:d})_j. \tag{3.8}$$

Thus, the learning objective for the model is to minimize the negative log-likelihood, which is equivalent to:

$$-\log(p_X(x)) = -\log(p_Y(f(x))) - \sum_j sc(x_{1:d})_j \tag{3.9}$$

This formulation allows the model to efficiently learn the transformation by leveraging the structure of the Jacobian, and it ensures that the latent space representation y = f(x) captures the complex distribution of the input data while maintaining the invertibility of the transformation.

3.3 Design and Implementation

We implement the Partial-State Conditioned Flow Adapter as described in Section 3.2. The core idea behind this flow adapter is to provide a flexible mechanism for transforming simple input distributions into more complex, signal-conditioned distributions, with the partial state s serving as the conditioning signal to guide these transformations (as shown in Figure 3.2). Unlike conventional flow models, where the transformation depends solely on the input vector, our design introduces the partial state s as an additional contextual cue, which helps steer the transformation process. Specifically, both the scaling function $sc(\cdot)$ and translation function $t(\cdot)$ take s as an input, thus conditioning the transformation on the agent's state at each time step.

The updated affine coupling layer transformations are as follows:

$$\begin{aligned} x_1' &= x_1 \\ x_2' &= x_2 \odot \exp(sc(x_1, s)) + t(x_1, s), \end{aligned} \tag{3.10}$$

where x_1 and x_2 represent the two subsets of the input vector. As illustrated in Figure 3.1, we relabel $x_{1:d}$ as x_1 and $x_{d+1:D}$ as x_2 for clarity and consistency. This relabeling is

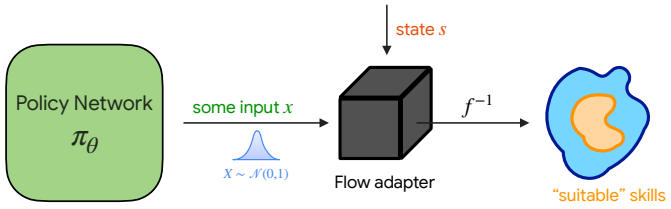


Figure 3.2: Design of flow adapter: input with simple distribution is then transformed to a complex distribution conditioned on the partial-state signal.

inspired by findings in the literature [46, 43] and corroborated by our own experimental results, which suggest that the composition of these subsets can significantly influence the overall learning process. To introduce variability and prevent overfitting, we apply a random permutation of the input vector between layers, this ensures that x_1 is not strictly the first d channels of the vector x, but rather a dynamically determined, randomly permuted subset that varies across layers during training. This randomness helps maintain robustness and adaptability in the flow adapter's performance.

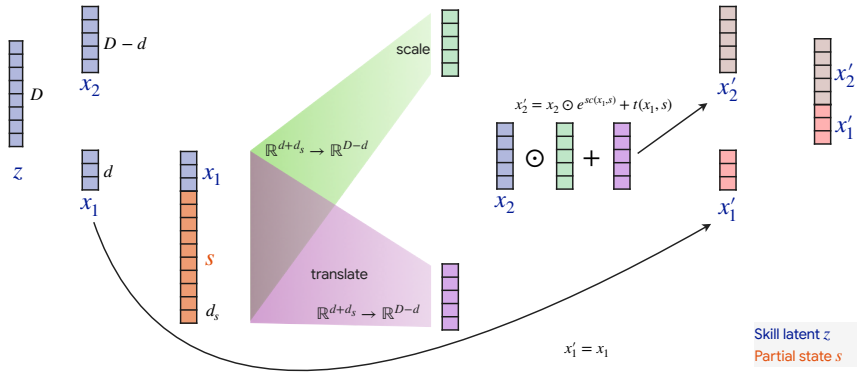


Figure 3.3: Implementation of coupling layer with partial state information as conditioning, in our implementation we use random permutation for breaking up the input vector z.

The specific details of a single coupling layer within the flow adapter are illustrated in Figure 3.3. In this design, the input to the coupling layer is the skill latent vector z, and the partial state vector is concatenated with the input vector segment before being passed to the scale and translation functions. This concatenation serves as a simple yet effective method for conditioning the transformation on state information. However, it is important to note that this operation between the latent vector and the partial state can be modified to employ more complex operations, depending on the desired level of conditioning or the specific requirements of the task at hand. This flexibility allows for customization and optimization based on the use case, making the flow adapter adaptable to a wide range of applications.

3.4 Training with Skill Latent Model

The Flow adapter can be trained concurrently with the skill latent model, as both have distinct training objectives and do not interfere with each other. When training the Flow adapter, the skill latent model operates in inference mode to generate the corresponding skill latent vector z for the state information s from the dataset. To maintain flexibility, the

skill latent model and the Flow adapter are trained in a single process, but in sequential order. The PyTorch like pseudo-code for this sequential training process is presented in Algorithm 1.

It is worth mentioning that the training logic for the flow adapter (lines 19 to 24) could be relocated or duplicated within the inner training loop of the skill latent model (after line 6). Further investigation may be required to evaluate the impact of this change on training dynamics and final results. However, to minimize computational costs and allow for customized training epochs for both the skill latent model and the flow adapter, we adopt a sequential training strategy.

Algorithm 1 Training Procedure for Skill Latent Model and Flow Adapter

Input: Training dataset with skill actions and states
Output: Trained Skill Latent Model and Flow Adapter

```
// Initialize variables
SL_epoch ← number of epochs for the skill latent model  FA_epoch ← number of epochs for the flow adapter
// Begin Skill Latent Model training
SLM.train()
for it ← 0 to SL_epoch do
    recon_loss ← 0  KL_loss ← 0  contrastive_loss ← 0
    for each batch in train_loader do
        { recon_input, μ, log_var, z } ← SLM.forward(batch)
        // Calculate reconstruction and KL losses
        recon_loss, KL_loss ← SLM.loss_function(recon_input, batch, μ, log_var)
        // Calculate contrastive loss
        contrastive_loss ← calculate_contrastive_loss(z, batch)
        // Total loss calculation
        loss ← recon_loss + β· KL_loss + λ· contrastive_loss
        // Backpropagation
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
    end
end
// Begin Flow Adapter training
FA.train()
for it ← 0 to FA_epoch do
    fa_loss ← 0
    for each batch in train_loader do
        // Perform SLM inference (no gradient)
        { recon, μ, log_var, z } ← SLM.forward(batch_noised, inference=True)
        // FA forward pass, conditioned on the first step of the state
        {log_p, log_jacob} ← FA.forward(z, state_input[...,0])
        // Calculate negative log likelihood
        loglikeloss ← (−log_p − log jacob).mean()
        // Backpropagation
        optimizer.zero_grad()
        loglikeloss.backward()
        optimizer.step()
    end
end
```

3.5 Experiments

In this section, we present the experimental results of training the flow adapter alongside the skill latent model, followed by a detailed visualization of the skill latent space before and after the flow adapter transformation. Specifically, we illustrate the distribution of the raw skill latent space generated by the skill latent model and how this distribution evolves once the flow adapter is applied.

3.5.1 Effects of Skill Latent Space Structure

As previously discussed, one of the benefits of training the Flow Adapter is that it does not interfere with the training of the skill latent model (SLM), they do not share loss functions, and the gradients of SLM are completely separated from the training of this flow adapter.

However, since the Flow Adapter operates on the skill latent vector z provided by the SLM, its task is to transform these latent vectors, conditioned on the partial state s, into a standardized form—specifically, a standard Gaussian distribution in our case. Given this dependence, we re-evaluate the hyperparameters used in training the skill latent model (as outlined in Section 2.4) to investigate how they influence the training dynamics of the Flow Adapter, allowing us to gain deeper insights into the overall system's behavior.

Sweep of the KL Regularization Term β. In this experiment, we investigate how different values of the KL regularization term β influence the learning process of the Flow Adapter. The primary evaluation metric used is the training loss of the Flow Adapter, defined as the negative log-likelihood, as described in Section 3.2. By adjusting β, we aim to understand its role in structuring the skill latent space and how this, in turn, affects the Flow Adapter's ability to perform its transformations. The results are visualized in Figure 3.4, which shows the relationship between Flow Adapter loss and β.

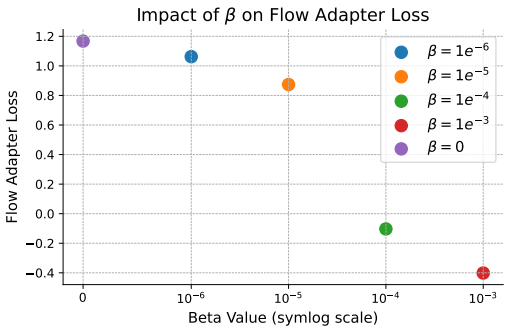


Figure 3.4: Flow adapter loss w.r.t β values.

The results indicate a correlation between the strength of KL regularization in the skill latent space and the negative log-likelihood loss of the Flow Adapter. Within the tested range of β values, where reconstruction quality is not significantly compromised, larger β values lead to lower Flow Adapter loss, suggesting that stronger regularization improves the adapter's performance. Together with the results we got from the experiment in skill latent model training, we can pick either $\beta = 1 \times 10^{-5}$ or $\beta = 1 \times 10^{-6}$.

Sweep of the Contrastive weight λ. In this experiment, we retained the same training configuration as in previous tests while fixing the β regularization parameter to 1×10^{-5}. Additionally we select temperature τ as 0.5. We then systematically swept through different values of the contrastive weight λ in the range [0, 5], evaluating the flow adapter loss for each setting. The results are visualized in Figure 3.5.

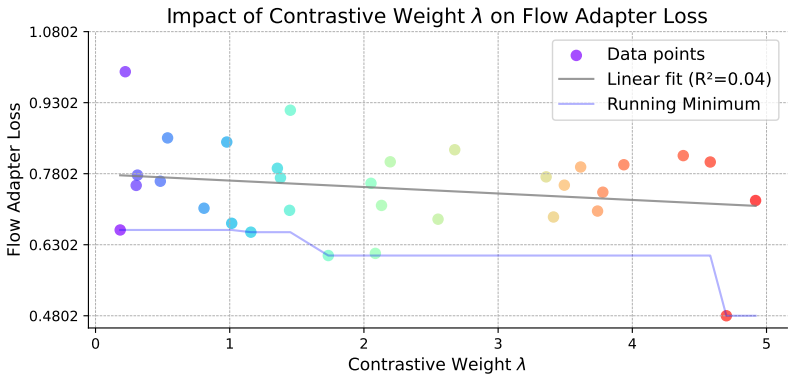


Figure 3.5: Flow adapter loss w.r.t contrastive weight.

As observed in the plot, the R^2 value of the linear regression fit is 0.04 which is close to zero, indicating an almost negligible correlation between the contrastive weight λ and the flow adapter loss. This suggests that the strength of the contrastive loss, does not have a straightforward linear relationship with the performance of the flow adapter. This is not entirely unexpected, given that contrastive learning can introduce variability depending on the specifics of the training data and batch shuffling.

One noteworthy observation is the presence of datapoints that lie on the running minimum curve. These points suggest that, at certain values of λ, the flow adapter loss can be reduced below that of the setup with no contrastive loss at all. Specifically, the minimum flow adapter loss decreases from $\sim$0.8 in Figure 3.4 (where no contrastive loss is applied) to $\sim$0.48 when contrastive learning is introduced, as shown in Figure 3.5. This indicates that contrastive learning, when appropriately balanced, can enhance the performance of the flow adapter by improving the representation of the latent space.

Visualization of skill latent space and transformed latent space. The transformation is denoted as $y = f(z, s)$, where each pair of latent vector z and partial state s is fed into the affine transformation function learned by flow adapter. Furthermore, we investigate how this latent space adapts when conditioned on the agent's current state, providing deeper insights into how the policy interacts with and explores this latent space during the policy learning process.

The left plot in Figure 4.11 shows the PCA result of the original distribution of the skill latent space before applying the flow adapter. Despite regularization through the β coefficient (as outlined in Chapter 2), the raw latent space remains relatively unstructured (as oppose to standard Gaussian distribution when KL regularization term is large). While there is some observable separation between the latents associated with different motions, there is also a degree of overlap, suggesting that certain skills share similarities in their abstract representations. This observation aligns with the research assumption that inherently distinct skills can still exhibit commonalities in their representations.

In contrast, the right plot in Figure 4.11 illustrates the skill latent space after the flow adapter has been applied. Here, the input state s acts as a conditioning signal for the transformation of each individual z as suggested by the training setup in Section 3.3. The resulting latent space is significantly more structured, displaying characteristics of a Gaussian distribution. This transformation not only refines the latent space but also introduces a more organized structure, which is crucial for enabling the policy to effectively navigate and utilize the latent space during learning. By conditioning on the current state of the agent, the flow adapter provides a more context-aware representation of the latent skills, aiding in the selection and application of appropriate skills for a given task.

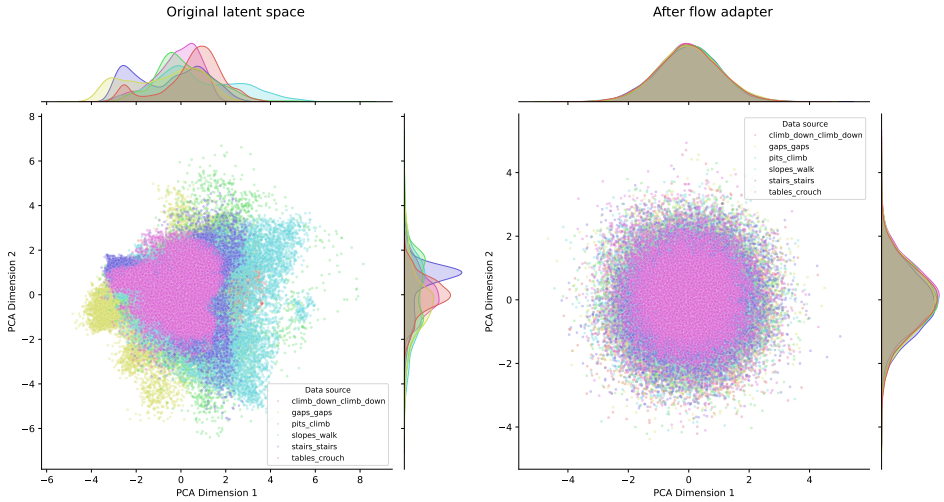


Figure 3.6: Visualization of the skill latent space using PCA. Left: original skill latent space; Right: skill latent space after flow adapter transformation: $y = f(z, s)$, here z and s are taking in pairs, which means latent vector is produced by the skill action a with the encoder of skill latent model, and z is the corresponding partial state of a.

3.6 Summary

This chapter introduces the Partial State-conditioned *Flow Adapter*, designed to enhance skill utilization during policy learning. This adapter learns a bijective mapping between the skill latent space and a standardized action space, conditioned on the agent's partial state, facilitating efficient exploration and context-aware skill selection. Leveraging Real NVP, the Flow Adapter utilizes invertible affine coupling layers parameterized by state-conditioned scale and translation functions.

Lightweight in nature, with $\sim 6\%$ number of parameters than the skill latent model, and can be trained concurrently with the skill latent model in a self-supervised manner, it transforms the unstructured latent space into a more organized Gaussian-like distribution without sacrificing generative quality. Experiments demonstrate that slight KL regularization of the skill latent space improves Flow Adapter training. While the impact of contrastive loss is less pronounced, it suggests potential benefits at specific values. Visualizations confirm the adapter's ability to structure the latent space, making it more suitable for policy exploration and skill selection during learning. This chapter establishes the Flow Adapter as a key component for enabling contextually appropriate and efficient skill utilization in subsequent policy learning stages.

Chapter 4

Policy Learning of Continuous Skill Learning Framework

Up til this point, we have introduced the two major components of the continuous skill learning framework described in Section 1.3: the skill latent model (in Chapter 2), and the flow adapter (in Chapter 3), now in this section we show how to build the Skill Latent Policy with these two components. Further we show how to build a relatively small policy network that builds upon the idea of refinement of the previously learned skills.

4.1 Policy Learning Formulation

In the domain of robotics, Reinforcement Learning (RL) has emerged as a powerful tool, particularly well-suited for scenarios where problems are naturally framed as sequential decision-making processes. At its core, RL leverages the Markov Decision Process (MDP) to formalize the control problem, typically represented as a tuple $(\mathcal{S}, \mathcal{A}, p, r, \gamma)$. Here, $\mathcal{S}$ denotes the state space, $\mathcal{A}$ represents the action space, $p(s_{t+1}|s_t, a_t)$ is the state transition function, $r(s_t, a_t, s_{t+1})$ defines the reward function, and γ is the discount factor which governs the importance of future rewards. In this context, a policy $\pi(a_t|s_t)$ describes the agent's behavior by mapping the current state s_t to a distribution over actions a_t. As the agent interacts with the environment, it generates a trajectory of states and actions, denoted by $\tau = (s_0, a_0, s_1, a_1, \dots)$. The probability distribution over these trajectories, given a policy π, is denoted as $\rho_\pi(\tau) = p(s_0) \prod_t \pi_t(a_t|s_t) p(s_{t+1}|s_t, a_t)$. The central objective in policy learning is to discover an optimal policy π^* that maximizes the sum of expected rewards across a trajectory. Formally, the objective function is given by:

$$J(\pi) = \mathbb{E}_{\tau \sim \rho_\pi} \left[\sum_{t=0}^{T} r(s_t, a_t) \right], \tag{4.1}$$

where $r(s_t, a_t)$ is the reward received at each time step t, and the expectation is taken over the distribution of trajectories induced by the policy π.

In robotic locomotion, particularly for tasks that involve continuous control and complex state-action spaces, on-policy model-free algorithms such as Trust Region Policy Optimization (TRPO) [47] and Proximal Policy Optimization (PPO) [48] have emerged as preferred choices. These algorithms are favored because they balance exploration and exploitation while ensuring stability during training. In all policy trainings of our work we use the default PPO implementation from [26] by only adding the weights initialization.

4.2 Components

In typical quadruped policy learning, there are several factors that are necessary and critical to the final performance either in simulation or in reality.

Simulator. A simulator is crucial for modeling the robot's interactions with its environment, enabling safe and scalable data collection before real-world testing. In this work, Isaac Gym [49] is used for its ability to run large-scale, parallel simulations efficiently.

Observation. The observation space in a Markov Decision Process (MDP) is critical, as it includes all sensory inputs, such as proprioception (e.g., joint positions, velocities) and exteroception (e.g., height scanner, LiDAR). These inputs represent the robot's state and environmental context. Observations for this work is introduced later in this chapter.

Reward. The reward function is key to shaping the desired behavior in an RL algorithm, as it sets the agent's objective. A typical approach is to design the reward function as a linear combination of different reward and penalty terms. In the task tested in the experiment section of this chapter, most rewards consists of a combination of tracking rewards (position, heading), rewards for limiting the action to not be too drastic and reduce collision (details in the appendix).

Curriculum. Curriculum learning involves progressively increasing the difficulty of tasks during training, much like how educational/gaming systems introduce more complex concepts/tasks over time. This approach helps agents tackle problems that are too challenging for traditional learning methods. We use the same game-inspired curriculum design from [26] where success leads to promotion to a higher difficulty, while failure results in demotion to an easier level. Robots mastering the highest difficulty are cycled back to random levels to maintain level diversity and mitigate catastrophic forgetting..

4.3 Policy Learning Framework

4.3.1 Skill Latent Policy

The skill latent policy operates by navigating within a skill latent space, rather than directly controlling the quadruped's joint positions as traditional control methods do. Instead of outputting joint positions for each of the quadruped's joints, the policy produces a set of parameters, denoted as y, that reside within the skill latent space. These parameters are subsequently decoded into joint actions by the skill latent decoder, which is trained according to the procedure outlined in Section 2.3.

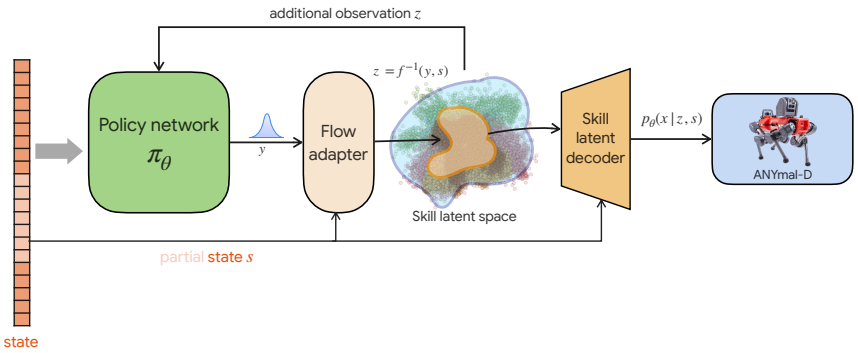


Figure 4.1: Overview of the skill latent policy. During policy inference, skill latent parameters from the previous time step are fed back into the policy as inputs for the current time step.

A key challenge in developing a skill latent policy is that, as the number of encoded skills within the latent space grows, efficiently searching through and navigating the latent space becomes increasingly important. To address this, we proposed in previous chapters to: 1) maintain the structure in the skill latent space. 2) introduce the flow adapter, which aids in directing the search toward regions of the latent space that are more relevant to the current state, thereby helping the policy identify the appropriate skills, as depicted in Figure 4.1, where the highlighted regions indicate the guiding aid provided by flow adapter. Therefore, when using the flow adapter, the parameters are first processed through it before being utilized in the skill latent space. Specifically, the parameters undergo an inverse transformation by the flow adapter, f^{-1}, which is conditioned on the current partial state s, yielding $f^{-1}(s, y)$. The resulting skill snippet actions for the current time step, $\mathbf{a}_t = (\hat{a}_1^t, \ldots, \hat{a}_H^t)$, are then generated by the skill latent decoder. The pytorch like code for the entire process is shown below.

Listing 4.1: Example of skill latent policy decoding.

```
latent = skill_latent_policy.act(obs, critic_obs)

partial_state = get_partial_state(obs)

if use_flow_adapter:
    z = flow_adapter.inverse(z, partial_state)

skill_snippet = skill_latent_model.decode(z, partial_state)
```

4.3.2 Learned Action Blending

The skill latent policy directly outputs parameters that can be decoded into an action sequence with a maximum length defined by the skill horizon H. This is conceptually similar to action chunking used in neuroscience [50, 51] and robotics manipulation tasks [52], where actions are grouped and executed as a unit, optimizing execution efficiency. However, our framework introduces two distinct features:

- We utilize recurrent models capable of decoding action sequences of varying lengths, as long as they fall within the training horizon H. This provides flexibility in generating actions that adapt to different tasks.
- The skill latent policy is executed at each time step, allowing for a flexible or learnable blending scheme that integrates current and past action sequences.

A learnable parameter, α_t, enables the blending of the current decoded action with previously generated actions. This approach helps produce smoother transitions between actions, which is particularly beneficial for real-world applications, as indicated by [51]. In Section 4.3.3, we demonstrate that this blending can be learned through lightweight policy optimization. The design is illustrated in Figure 4.2. In this design, the output action at time step $t = 4$ is computed as:

$$\alpha_4 d_1 + (1 - \alpha_4)(\alpha_3 c_2 + (1 - \alpha_3)(\alpha_2 b_3 + (1 - \alpha_2)a_4)),$$

where α_t is the blending weight generated by the refinement policy at each step, ensuring smooth transitions between actions over time.

4.3.3 Refinement Policy

The skill latent policy is designed to utilize previously learned "muscle memories" to accelerate the learning of new tasks. However, when a task demands actions that deviate

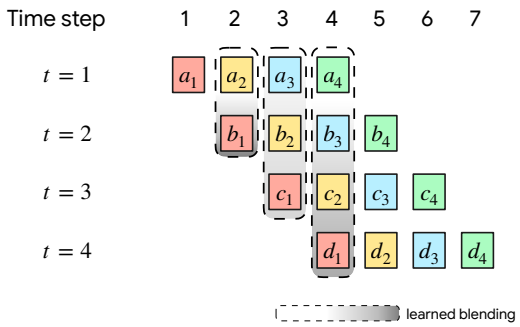


Figure 4.2: Learned action blending: each action is blended with a coefficient α_t with the actions generated at previous time steps.

significantly from those encoded in the skill latent space or requires fine-tuning (such as refining the relative scale between timesteps to cater to dynamic environment), the refinement policy is introduced.

The refinement policy shares the same architecture as the skill latent policy, but differs in its training sequence, action space, and how its output is utilized. It is trained after the skill latent policy and is responsible for two key elements: a blending weight α_t (discussed in Section 4.3.2) and a residual action a^t_Δ. These outputs enable the refinement policy to make adjustments to the actions generated by the skill latent policy, allowing for more precise control and adaptation to new environments or task variations. Training for the refinement

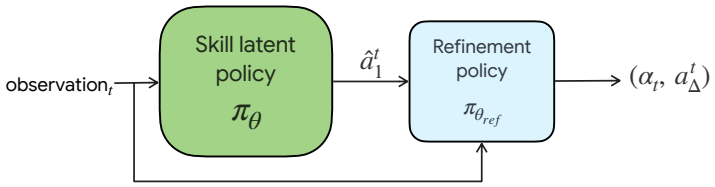


Figure 4.3: Observation and output of refinement policy $\pi_{\theta_{ref}}$. It takes the same variables of observation as the skill latent policy but at the time t when the refinement policy is called, together with the first step (current step) output from skill latent policy π_θ, the output is a blending weight α_t and a residual action at current step a^t_Δ.

policy begins after the skill latent policy has undergone a sufficient number of iterations. In practice, we initiate this process once the terrain level performance plateaus. At that point, the weights of the skill latent policy are frozen, treating it as a fixed, parameterized policy that maps input observations to actions. The refinement policy then adjusts these actions via the blending weight α_t, enabling fine-tuning without disrupting the core learned actions.

4.4 Configuration

We show the configuration of observation space in Table 4.1 and based on the results we got from the experiments in Chapter 2 and Chapter 3 we took the S4D version skill latent model checkpoint from the configuration of $\beta = 1 \times 10^{-5}$, contrastive weight $\lambda = 1.73$, temperature $\tau = 0.5$ and latent dimension $d = 10$.

The configuration for skill latent policy is shown in Table A.1, and configuration for refinement policy is shown in Table A.2

Component	Notation
Base linear velocity	v_b
Base angular velocity	ω_b
Projected gravity	g_b
Position command	p_t^c
Heading command	ψ_t^c
Time left to target	t_G^*
Joint positions and velocities	q_t, $\dot{q}_t$
Decoded joint command of previous step	a_{t-1}
Heightmap scan ($2m \times 1m$)	h
Skill latent of previous step	z_{t-1}

Table 4.1: Observation components and their corresponding notations.

4.5 Testing on Multiple Terrains

We evaluate the proposed framework on two categories of tasks, also referred to as *terrains*: (1) Tasks where the skills were encoded during the training phase of the skill latent space, requiring efficient exploration within this latent space; and (2) Tasks involving unseen terrains, which were not encountered during training of skill latent model and require not only exploration but also recombination and refinement of learned skills. Table B.2 provides a detailed overview of the terrains used in the experiments.

In the first experiment section, we present a performance comparison between the proposed continuous skill learning framework (in this section we only test the skill latent policy) and a baseline method. Each experiment is run with 5 different seeds and deterministic training enabled where in each curve the shaded area represents the max and min value, with the solid line representing the mean value. The baseline method directly outputs joint target positions, whereas our approach generates skill latent parameters, which are then decoded into joint positions (referred to as actions). To evaluate performance, we use several key metrics, including learning curves for multiple reward functions, terrain difficulty levels, and success rates across different terrains. The evaluation follows a curriculum-based setup, where the robot progressively encounters increasingly challenging terrains. Success rates are measured at each difficulty level to assess the framework's ability to adapt, recombine, and generalize to each environments. By comparing these metrics, we aim to evaluate the efficiency, adaptability, and robustness of our continuous skill learning framework relative to the traditional method.

Gaps. We compare the performance of our approach (skill latent policy) against the baseline method in terms of terrain level progression on gaps terrain, using the same curriculum, reward setup, and command types as the baseline. The results, shown in Figure 4.4, indicate that our method achieves faster initial learning compared to the baseline. In this context, terrain level refers to the average level across all parallel training agents. Initially, agents are randomly assigned to one of the lower terrain levels. As training progresses, agents that successfully complete a level (short of the maximum) are promoted to a higher level, while those that fail are moved to a lower level. If an agent succeeds on all levels, it is randomly reassigned to a new level. A higher terrain level in the learning curve indicates better overall performance, as the policy is capable of handling more difficult terrains. The small overshoots in the graph reflect random level assignments after reaching the highest level. Additionally, we compare both methods on a set of rewards that penalize excessive physical motions, including torque limits, joint velocity limits, contact forces, and collisions. The comparison is illustrated in lower four figures of Figure 4.4. The visualizations show that the actions learned through our skill latent policy are smoother and more controlled, resulting in higher rewards related to maintaining torque and velocity within acceptable

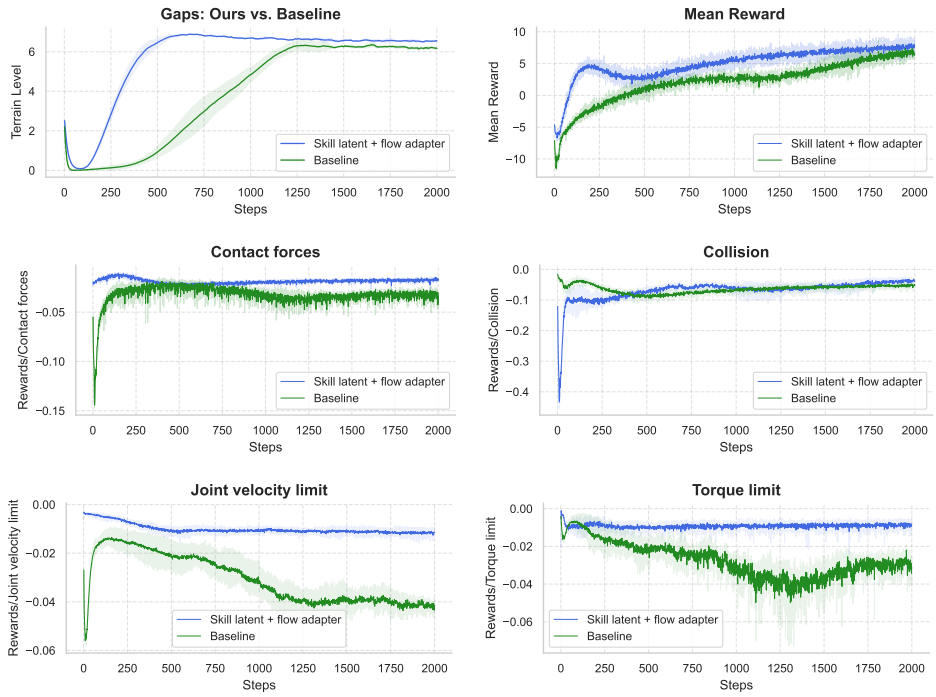


Figure 4.4: Performance comparison between our method (skill latent policy) and the baseline on the gaps terrain. Metrics include terrain level progression, mean reward, and physical penalties such as collision, contact forces, joint velocities, and torque limits.

limits. This demonstrates that our approach not only accelerates learning but also leads to more gentle behaviors.

Table crouching A similar evaluation was conducted for the table crouching task, with results shown in Figure 4.5. As with the gaps task, our method outperforms the baseline, showing faster learning progression on this task as well. We also evaluated both methods based on physical penalties for the table crouching task, including limits on contact forces, joint velocities, torque, and collisions. These results are presented in Figure 4.5. Agreeing with the previous evaluation on type 1 task.

Climb up box. The results in Figure 4.6 show that the skill latent policy significantly outperforms the baseline early in the task training. The terrain level rises rapidly, reaching a plateau around 500 steps, which is mirrored in the mean reward plot, where the skill latent policy achieves slighly higher rewards from the beginning and maintains a considerable gap over the baseline throughout the task. In terms of contact forces and collisions, the skill latent policy demonstrates better performance, the baseline performs better in the early stages with respect to joint velocity and torque limits, the skill latent policy eventually surpasses the baseline as training progresses, leading to more stable and efficient control later in the task.

Obstacles. We show the results on obstacles terrain in Figure 4.7, comparing to terrain Gaps and Table, the skill latent policy still reaches its rise up moment at around 500 steps while the baseline methods also reaches around the same time with relatively lower average terrain level, this curves identifies that both methods learn relatively fast to gain up to higher

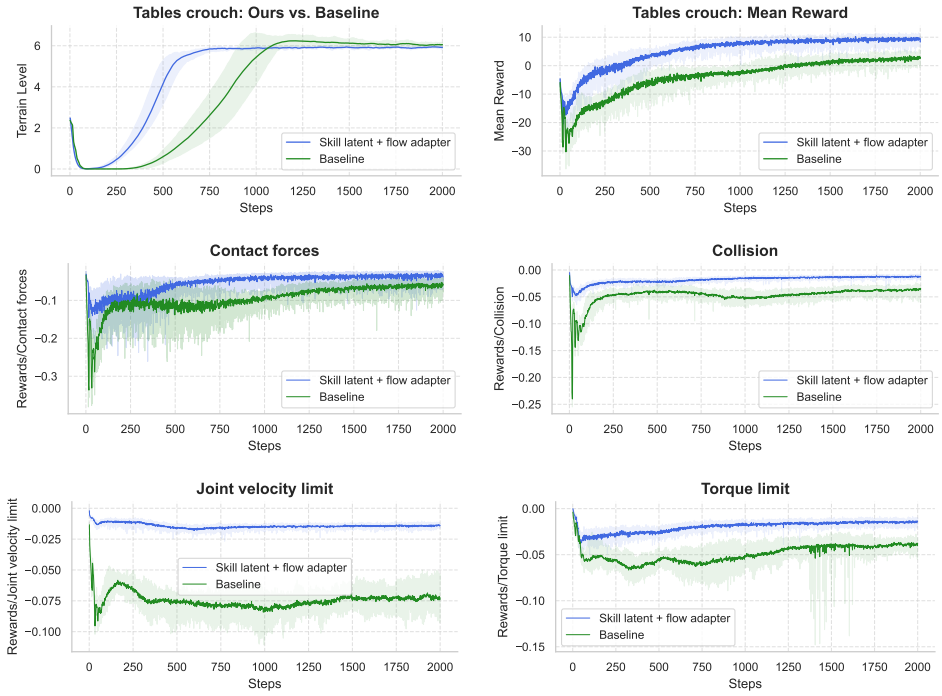


Figure 4.5: Evaluation results for the table crouching task, comparing terrain level, mean reward, and physical penalties such as contact forces, collisions, joint velocities, and torque limits between our method and the baseline.

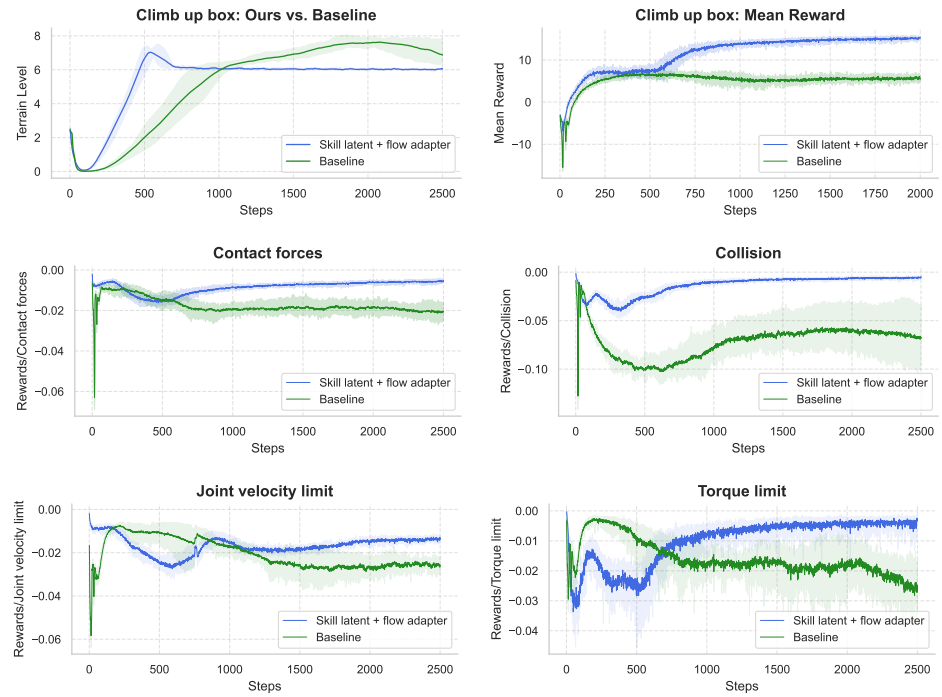


Figure 4.6: Results for the “Climb Up Box” task. The plots show terrain level progression, mean reward, and physical penalty metrics (contact force, collision, joint velocity, and torque limits) for both the baseline and skill latent policy.

level of terrains, signifying the difficulty of this terrain might not lies in the traversability aspect but more on the planning aspect.

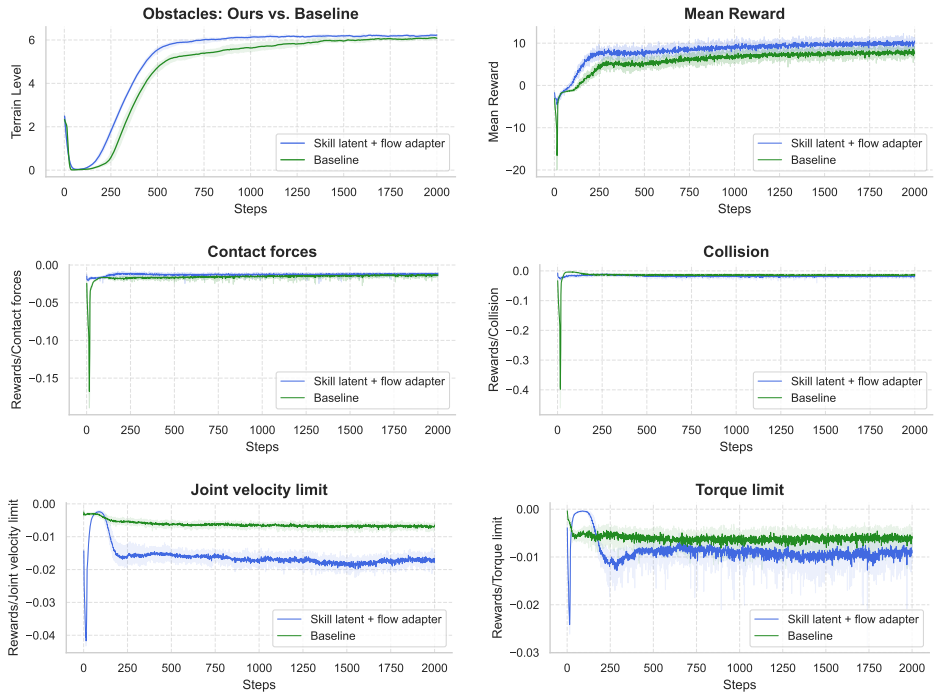


Figure 4.7: Terrain level, mean reward, and physical penalties for the obstacles terrain. The results show the comparison between the baseline method and our skill latent policy.

Rock piles We test both methods on rock piles terrain and report the results in Figure 4.8, the results for the rock pile terrain indicate that the skill latent policy with the flow adapter does not show a significant advantage over the baseline, except in a few metrics. Both methods demonstrate similar performance in terms of terrain level progression, with both reaching a plateau around step 500. However, the skill latent policy does show a slightly faster learning speed early on, achieving a faster terrain progression within the first 200-500 steps. Specifically, skill latent policy maintains slightly lower joint velocities and torque values, which contribute to more gentle motions and slightly higher rewards in these areas. Overall, while the skill latent policy provides some early learning advantages and results in gentler motion, the performance gap between it and the baseline is minimal on this terrain.

Simple Parkour. We evaluated the method on a simple parkour task, as detailed in Table B.2, and reported the same metrics used in previous evaluations, as shown in Figure 4.9. Notably, this terrain falls under type 2, meaning that during training, the skill latent model and flow adapter did not encounter actions collected from similar tasks or terrains. Despite this, the skill latent policy demonstrates a rapid rise in terrain level within the first 250 steps, indicating the agent's ability to adapt quickly to the new task. This is further reflected in the reward, which surpasses 0 within the first 100 steps. In terms of contact forces and collisions, both methods exhibit comparable performance. As expected, the skill latent policy leads to higher rewards when it comes to joint velocity and torque limit, highlighting its ability to maintain smoother and more efficient movements.

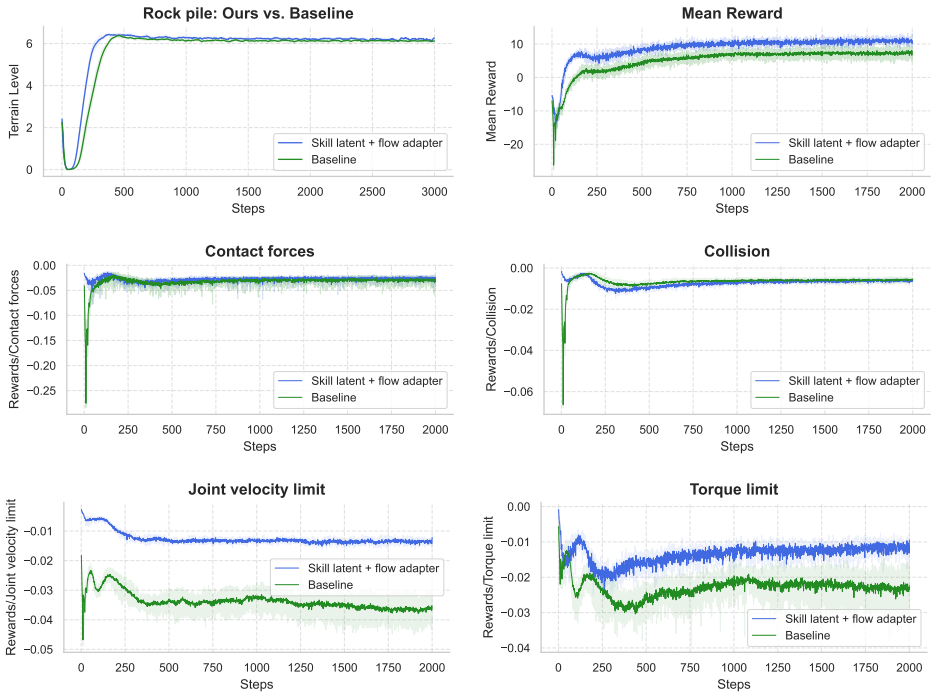


Figure 4.8: Terrain level, mean reward, and physical penalties for the rock pile terrain. The results show the comparison between the baseline method and our skill latent policy.

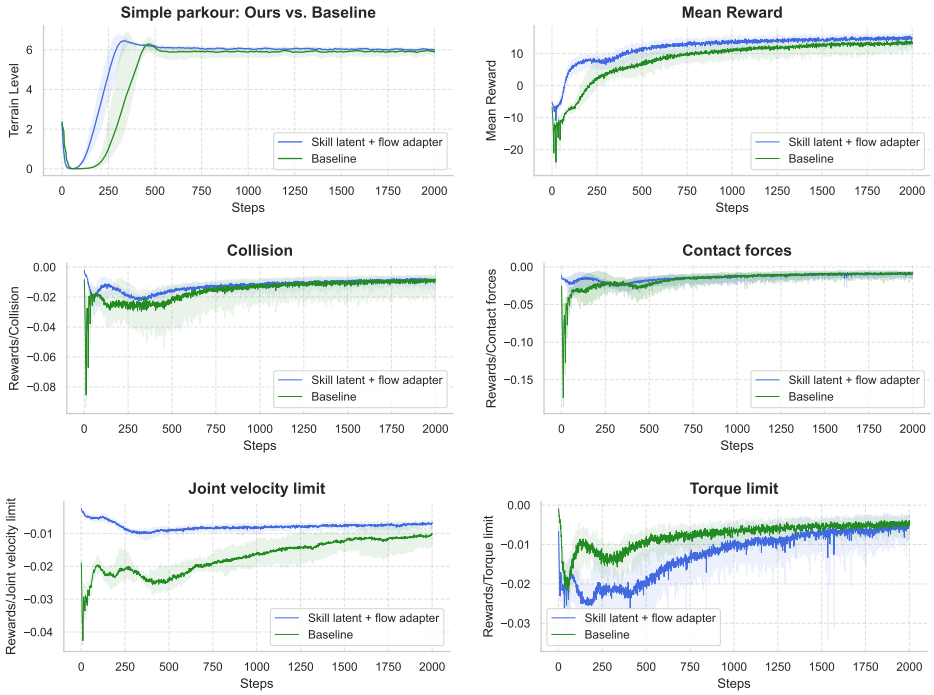


Figure 4.9: Terrain level, mean reward, and physical penalties for the simple parkour terrain. The results show the comparison between the baseline method and our skill latent policy.

4.6 Analyses and Studies

How Does the Flow Adapter Help? We present a case study illustrating the role of the flow adapter in improving skill learning. In Figure 4.10, we compare the performance difference between our approach and baseline, in our approach we ablate the choice of using and not using flow adapter for policy learning.

The objective is to examine how the policy selects actions in the skill latent space during the early stages of policy learning, as it begins to explore this space and how flow adapter affects this process. For notations, each action, denoted as a_i, has a corresponding initial partial state s_i, and we assume the effectiveness (how likely it can be used as an effective skill at this context) of a_i in a given context is characterized by a partial state s_{sample}, can be inferred by measuring the similarity between s_i and s_{sample}. The relevance of action a_i to the sampled state s_{sample} is quantified using cosine similarity, the cosine similarity is defined as:

$$\text{sim}(s_i, s_{\text{sample}}) = \frac{s_i \cdot s_{\text{sample}}}{|s_i||s_{\text{sample}}|}$$

where $|\cdot|$ denotes the Euclidean norm, and $\cdot$ represents the dot product between the two state vectors. A higher cosine similarity between s_i and s_{sample} suggests that a_i is more contextually relevant, increasing the likelihood that it will result in effective action when executed in the current state s_{sample}.

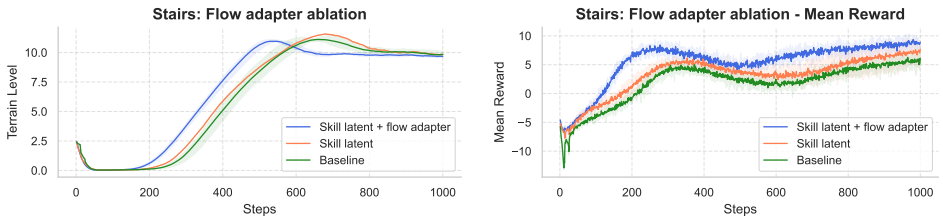


Figure 4.10: Performance comparison on stairs terrain between the skill latent policy with and without the flow adapter. The results show terrain level progression and mean reward.

In the following two figures, we visualize the distribution of actions within the skill latent space using Principal Component Analysis (PCA). States with cosine similarity greater than 0.7 (chosen as $\cos \pi/4$) to the sampled state s_{sample} are highlighted in red. The sampled state, s_{sample}, is drawn from training data collected on a similar stairs terrain. The left figure shows that even after applying PCA, the original skill latent vectors remain dispersed due to the inherent structure of the skill latent space, despite regularization. In contrast, the right figure illustrates the effect of applying the flow adapter, which transforms each skill latent vector z_i in the training dataset based on the sampled state: $y_i = f(s_{\text{sample}}, z_i)$. The transformed latent space exhibits a more structured distribution, which enhances the policy's ability to explore efficiently and select relevant skills in the current context. The results demonstrate that the fast learning speed of the flow adapter can be attributed to its ability to create a more structured latent space, making it easier for the policy to explore and select actions that correspond more effectively to the current context.

When does this framework tend to fail? The skill latent model is designed to encode past skill snippet actions into a continuous latent space, with a parameter-efficient flow adapter guiding the policy toward the most relevant skill latents for the current state during exploration. While this approach offers computational efficiency and enhanced guidance, it is not without its limitations. One key trade-off lies in the potential to overlook other useful skill latents or restrict exploration to a limited variety of skill snippets within the latent space. This can occur if the offline dataset/buffer used to train/finetune the flow adapter

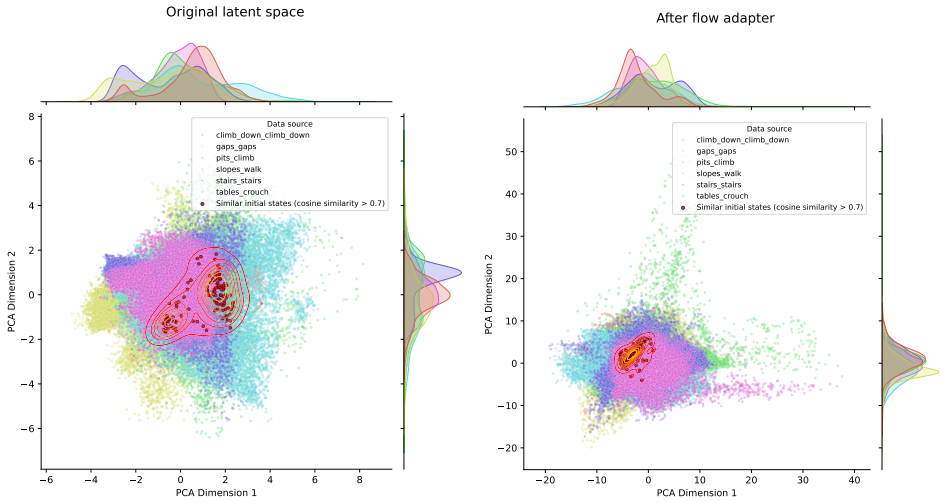


Figure 4.11: Visualization of the skill latent space using PCA. Left: original skill latent space; Right: latent space after applying the flow adapter. The flow adapter creates a more structured latent space, making it easier for the policy to explore relevant actions.

lacks diversity, resulting in gaps in the state, action pairs that could be critical for adapting to new tasks or environments. In such cases, the policy may become overly dependent on the flow adapter's guidance, causing it to focus on suboptimal regions of the latent space and neglect better-performing regions. As a result, the policy risks getting stuck in less effective regions, which can impede optimal performance, particularly in dynamic and complex tasks.

To further explore these limitations, we conducted a case study using the *simple parkour* task. In this task, we increased the maximum box height from 0.35m to 0.5m, while keeping all other configurations unchanged (this modified task is referred to as "parkour"). We compared the baseline method with three variations of the skill latent policy: one without a flow adapter, one with a flow adapter trained for 20 epochs, and one with a flow adapter trained for 100 epochs (the default number we use for all previous experiments).

As demonstrated in Chapter 3, longer training durations for the flow adapter result in lower flow adapter (FA) loss, indicating stronger guidance within the latent space. The learning curves in Figure 4.12 show that the skill latent policy with a flow adapter trained for 100 epochs initially learns the fastest, as evidenced by a rapid rise in terrain level and mean reward. However, this rapid learning slows down between 200 and 600 steps, and ultimately, the terrain level plateaus around the same time as the baseline method.

In contrast, the version without the flow adapter shows a slower initial learning rate. Nevertheless, after the initial phase, it surpasses the performance of the guided versions. Without the flow adapter's influence, the policy explores the latent space more freely, leading to faster learning later on, and it is the first to reach the terrain level plateau. Furthermore, this unguided version achieves the highest values across all metrics, including mean reward and physical penalty constraints.

To further validate this observation, we evaluate these three setups across different terrain levels of the parkour terrain, including an out-of-distribution (OOD) level. The results are shown in Figure 4.13.

The results from this case study highlight that while the flow adapter provides valuable structure and guidance in early learning phases, it may limit exploration and, consequently, performance in the long run. This trade-off underscores the importance of

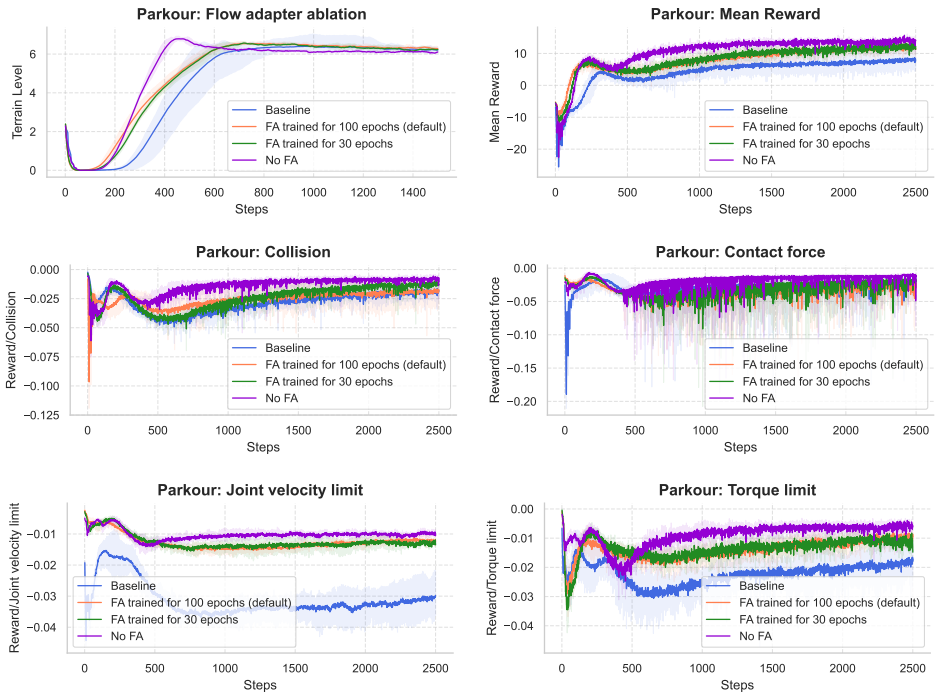


Figure 4.12: Performance comparison on the simple parkour task between skill latent policies with and without flow adapter. Metrics include terrain level, mean reward, and physical penalties such as contact forces, collisions, joint velocities, and torque limits.

Observation on refinement policy learning. During the second stage of training, where the skill latent policy is kept frozen and only the refinement policy is updated, we empirically observed that the parameter α_t decreases consistently throughout the training iterations. Additionally, the rewards associated with penalizing drastic actions increase as training progresses.

Naively Scaling Up the Skill Horizon. In all of our experiments, we used a fixed skill horizon length of $H = 5$. While this setting proved effective for short-horizon tasks, it inherently limits the agent's ability to learn and execute longer-horizon skills. A longer horizon could enable the agent to maintain consistent and coordinated actions over an extended period, which is particularly important for tasks requiring sustained control and precision, such as long horizon parkour or traversing complex, uneven terrains.

To explore the potential benefits of longer horizons, we increased the horizon length from 5 to 15 using the same framework and tested the performance on the parkour terrain (as shown in Figure 4.12). However, as depicted in Figure 4.14, both terrain level and mean reward performance declined when scaling up the horizon length. This unexpected result suggests that simply increasing the horizon without further adjustments to the framework may introduce challenges in learning longer-horizon skills.

The observed performance drop could be due to several factors, such as the lack of structure in the latent space, which may overwhelm the current policy during exploration. Additionally, the agent might require more sophisticated training strategies, such as curriculum learning, to gradually adapt to the demands of longer horizons. These results highlight the need for further investigation and refinement to fully leverage longer-horizon capabilities, a topic we will discuss in more detail in Chapter 5.

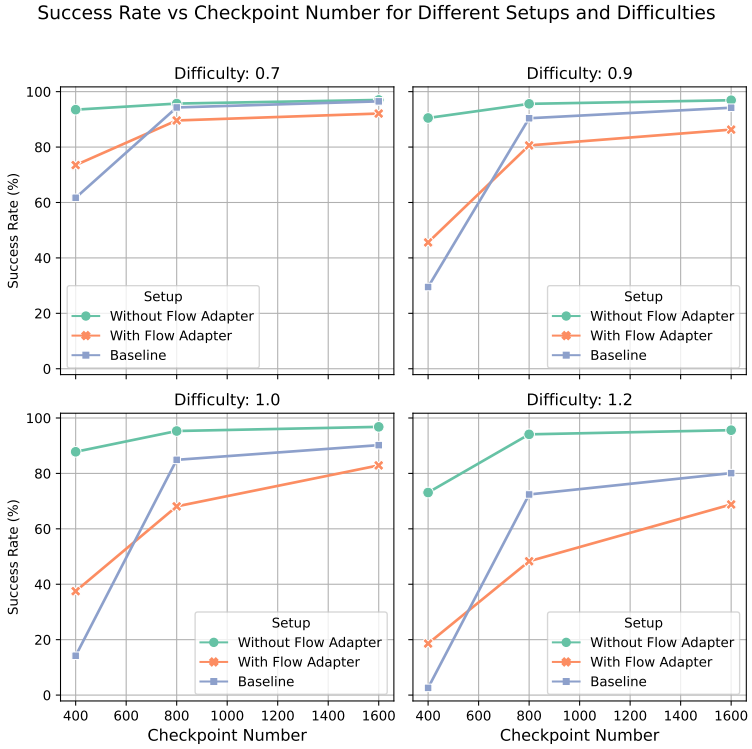


Figure 4.13: Parkour terrain: Success rate vs. checkpoint number for different terrain difficulty levels. The plot compares three methods: Without Flow Adapter, With Flow Adapter, and Baseline. Across all terrain difficulties, the setup without the flow adapter consistently achieves higher success, surpassing both the flow-guided and baseline versions. The out-of-distribution terrain (Difficulty 1.2) demonstrates a more significant gap, highlighting the case when it is benefitial to free explore in the latent space.

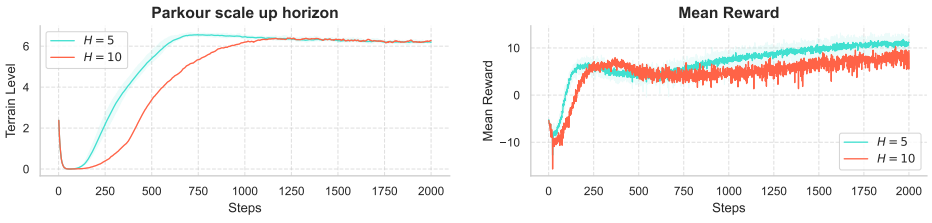


Figure 4.14: Comparison of terrain level and mean reward for the simple parkour task when scaling the horizon length from $H = 5$ to $H = 15$. The results demonstrate that increasing the horizon length without further adjustments to the framework leads to a decline in both terrain level and mean reward, highlighting the challenges associated with managing longer-horizon actions.

4.7 Summary

This chapter focuses on the policy learning component of the proposed continuous skill learning framework. First, the Skill Latent Policy is introduced, which operates within the learned skill latent space (Chapter 2) rather than directly generating joint actions. This policy outputs parameters that are decoded into action sequences by the skill latent model decoder, allowing the reuse of previously learned skill snippets. To improve exploration and skill selection within this latent space, the Flow Adapter (Chapter 3) is incorporated. This module guides the policy toward relevant regions of the latent space based on the current state, promoting efficient exploration and skill retrieval.

Additionally, the Refinement Policy is introduced to fine-tune the actions produced by the Skill Latent Policy. This enables the robot to handle terrains or tasks that require more precise control than the encoded skills can provide, this is achieved by a Learned Action Blending mechanism to ensure smooth transitions between actions generated at different time steps, ensuring fluid and continuous motion. Extensive evaluations on various terrains demonstrate that the proposed framework achieves faster or equivalent learning and smoother actions compared to a baseline method that directly outputs joint positions.

Analyses emphasize the importance of the Flow Adapter in structuring the latent space and facilitating efficient exploration. However, a case study reveals a potential trade-off: over-reliance on the Flow Adapter may limit exploration in some scenarios, suggesting the need for further research into its optimal use. Finally, the chapter discusses the challenges of scaling up the skill horizon length. The results show that merely increasing the horizon without adjusting the framework can hinder learning, underscoring the need for more sophisticated training strategies and latent space representations to effectively utilize longer-horizon skills. These findings provide valuable insights for future research, including optimizing the Flow Adapter's role and developing methods for robust learning with extended skill horizons.

Chapter 5

Conclusion

5.1 Summary

In this work, we presented a novel and effective framework for continuous skill learning on the ANYmal quadruped robot. The framework integrates key concepts such as skill decomposition, recombination, and refinement, enabling the robot to leverage previously learned skills and adapt them to new environments and tasks efficiently. Through the use of both latent space and low-level action space policies, we demonstrated how an agent can maintain adaptability and scalability when learning complex motor tasks.

What it achieves. The proposed framework provides a structured, yet flexible, approach to learning and refining skills in a continuous manner. It allows for the encoding of learned skills into a continuous latent space and facilitates the recombination of these skill snippets to form more complex behaviors. By incorporating a refinement policy, the system enables fine-tuning of these skills based on the specific requirements of new tasks, ensuring more precise control and higher overall performance.

Current limitations. While the framework has proven effective for skill learning with skill snippet of short horizons, it is not yet a substitute for hierarchical reinforcement learning, particularly when dealing with tasks that require significantly longer skill horizons. As the horizon length increases, the learning speed tends to decrease as shown in the previous chapter, especially in the early stages of training. This is a key limitation that affects the scalability of the system for long-horizon tasks.

5.2 Future Work

There is considerable potential to extend this framework by addressing the current limitations. For instance, future work could focus on scaling up the skill horizon length in the latent space and learning representations with the same fashion as introduced in this work but encoding skill snippets with different horizon lengths thanks to the recurrent nature of skill latent model. and during policy learning, curriculum learning can be designed to incentive the agents gradually learn to use longer horizon skills. Since the skill latent model is recurrent in nature, it is capable of encoding and decoding longer skill sequences without changing the structure of the model. Further research could also explore more advanced representation learning techniques to enhance the structure of the skill latent space for efficient exploration.

Additionally, this work introduced a lightweight flow adapter, which allows for efficient fine-tuning when new experiences are added to the system. Unlike the skill latent model,

which is parameter-heavy and computationally expensive to retrain, the flow adapter provides a flexible yet low-computation method for adjusting skill latents. This significantly reduces the computational overhead required for fine-tuning, as the flow adapter can modify or even rebalance the latent space to accommodate new tasks without the need to re-train the entire skill latent model. Future research could further explore the flow adapter's capabilities in dynamically adjusting to unseen environments and how it could be leveraged to continually refine learned skills with minimal computational cost.

Moreover, integrating hierarchical reinforcement learning strategies into the framework could help address the challenges of long-horizon tasks, without the need to always train with the correct horizon length for skill snippets in the skill latent model. By utilizing a shared latent space for actions, individual low-level policies can be trained more efficiently and at a faster pace, ultimately improving both learning speed and performance in complex, long-horizon environments.

Finally, this work is closely aligned with the principles of continual learning and continual reinforcement learning. The ultimate goal is to create a system capable of continuously learning and refining skills, adapting to new tasks without forgetting previously acquired knowledge. This continuous, evolving learning process is central to the broader objectives of continual learning research. Future work could explore how the framework can be extended to better support continual skill acquisition and refinement, particularly in lifelong learning scenarios where the agent is expected to encounter a wide range of tasks over time. This could involve developing mechanisms for efficient knowledge transfer between tasks, addressing catastrophic forgetting, and ensuring the scalability of the system as it encounters an increasing number of tasks and environments.

Bibliography

[1] D. Kudithipudi, M. Aguilar-Simon, J. Babb, M. Bazhenov, D. Blackiston, J. Bongard, A. P. Brna, S. Chakravarthi Raja, N. Cheney, J. Clune *et al.*, "Biological underpinnings for lifelong learning machines," *Nature Machine Intelligence*, vol. 4, no. 3, pp. 196–210, 2022.

[2] T. Mitchell, W. Cohen, E. Hruschka, P. Talukdar, B. Yang, J. Betteridge, A. Carlson, B. Dalvi, M. Gardner, B. Kisiel *et al.*, "Never-ending learning," *Communications of the ACM*, vol. 61, no. 5, pp. 103–115, 2018.

[3] D. Abel, A. Barreto, B. Van Roy, D. Precup, H. P. van Hasselt, and S. Singh, "A definition of continual reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[4] M. Hutter, C. Gehring, D. Jud, A. Lauber, C. D. Bellicoso, V. Tsounis, J. Hwangbo, K. Bodie, P. Fankhauser, M. Bloesch *et al.*, "Anymal-a highly mobile and dynamic quadrupedal robot," in *2016 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, 2016, pp. 38–44.

[5] S. Hochreiter, "Long short-term memory," *Neural Computation MIT-Press*, 1997.

[6] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder–decoder approaches," in *SSST@EMNLP*, 2014.

[7] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

[8] R. Pascanu, "On the difficulty of training recurrent neural networks," *arXiv preprint arXiv:1211.5063*, 2013.

[9] A. Vaswani, "Attention is all you need," *Advances in Neural Information Processing Systems*, 2017.

[10] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, J. Burstein, C. Doran, and T. Solorio, Eds. Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 4171–4186.

[11] T. B. Brown, "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*, 2020.

[12] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," Jun. 2021, arXiv:2010.11929 [cs].

[13] M. Caron, H. Touvron, I. Misra, H. Jégou, J. Mairal, P. Bojanowski, and A. Joulin, "Emerging properties in self-supervised vision transformers," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 9650–9660.

[14] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko *et al.*, "Highly accurate protein structure prediction with alphafold," *nature*, vol. 596, no. 7873, pp. 583–589, 2021.

[15] A. Gu, K. Goel, and C. Ré, "Efficiently modeling long sequences with structured state spaces," *arXiv preprint arXiv:2111.00396*, 2021.

[16] J. T. Smith, A. Warrington, and S. W. Linderman, "Simplified state space layers for sequence modeling," *arXiv preprint arXiv:2208.04933*, 2022.

[17] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, X. Chen, K. Choromanski, T. Ding, D. Driess, A. Dubey, C. Finn *et al.*, "Rt-2: Vision-language-action models transfer web knowledge to robotic control," *arXiv preprint arXiv:2307.15818*, 2023.

[18] B. Mitrevski, A. Rak, J. Schnitzler, C. Li, A. Maksai, J. Berent, and C. Musat, "Inksight: Offline-to-online handwriting conversion by learning to read and write," *arXiv preprint arXiv:2402.05804*, 2024.

[19] A. Chaudhry, M. Rohrbach, M. Elhoseiny, T. Ajanthan, P. K. Dokania, P. H. Torr, and M. Ranzato, "On tiny episodic memories in continual learning," *arXiv preprint arXiv:1902.10486*, 2019.

[20] K. Pertsch, Y. Lee, and J. Lim, "Accelerating reinforcement learning with learned skill priors," in *Conference on robot learning*. PMLR, 2021, pp. 188–204.

[21] K. Rana, M. Xu, B. Tidd, M. Milford, and N. Sünderhauf, "Residual skill policies: Learning an adaptable skill-based action space for reinforcement learning for robotics," in *Conference on Robot Learning*. PMLR, 2023, pp. 2095–2104.

[22] K. Hausman, J. T. Springenberg, Z. Wang, N. Heess, and M. Riedmiller, "Learning an embedding space for transferable robot skills," in *International Conference on Learning Representations*, 2018.

[23] A. Iscen, K. Caluwaerts, J. Tan, T. Zhang, E. Coumans, V. Sindhwani, and V. Vanhoucke, "Policies modulating trajectory generators," in *Conference on Robot Learning*. PMLR, 2018, pp. 916–926.

[24] T. Haarnoja, B. Moran, G. Lever, S. H. Huang, D. Tirumala, J. Humplik, M. Wulfmeier, S. Tunyasuvunakool, N. Y. Siegel, R. Hafner *et al.*, "Learning agile soccer skills for a bipedal robot with deep reinforcement learning," *Science Robotics*, vol. 9, no. 89, p. eadi8022, 2024.

[25] D. Hoeller, N. Rudin, D. Sako, and M. Hutter, "Anymal parkour: Learning agile navigation for quadrupedal robots," *Science Robotics*, vol. 9, no. 88, p. eadi7566, 2024.

[26] N. Rudin, D. Hoeller, P. Reist, and M. Hutter, "Learning to walk in minutes using massively parallel deep reinforcement learning," in *Conference on Robot Learning*. PMLR, 2022, pp. 91–100.

[27] X. B. Peng, Y. Guo, L. Halper, S. Levine, and S. Fidler, "Ase: Large-scale reusable adversarial skill embeddings for physically simulated characters," *ACM Transactions On Graphics (TOG)*, vol. 41, no. 4, pp. 1–17, 2022.

[28] H. Yao, Z. Song, B. Chen, and L. Liu, "Controlvae: Model-based learning of generative controllers for physics-based characters," *ACM Transactions on Graphics (TOG)*, vol. 41, no. 6, pp. 1–16, 2022.

[29] J. Merel, L. Hasenclever, A. Galashov, A. Ahuja, V. Pham, G. Wayne, Y. W. Teh, and N. Heess, "Neural probabilistic motor primitives for humanoid control," *arXiv preprint arXiv:1811.11711*, 2018.

[30] Z. Luo, J. Cao, J. Merel, A. Winkler, J. Huang, K. Kitani, and W. Xu, "Universal humanoid motion representations for physics-based control," *arXiv preprint arXiv:2310.04582*, 2023.

[31] S. Starke, I. Mason, and T. Komura, "Deepphase: Periodic autoencoders for learning motion phase manifolds," *ACM Transactions on Graphics (TOG)*, vol. 41, no. 4, pp. 1–13, 2022.

[32] C. Li, E. Stanger-Jones, S. Heim, and S. Kim, "Fld: Fourier latent dynamics for structured motion representation and learning," *arXiv preprint arXiv:2402.13820*, 2024.

[33] Y. Tay, M. Dehghani, S. Abnar, Y. Shen, D. Bahri, P. Pham, J. Rao, L. Yang, S. Ruder, and D. Metzler, "Long range arena : A benchmark for efficient transformers," in *International Conference on Learning Representations*, 2021.

[34] A. Gu, I. Johnson, A. Timalsina, A. Rudra, and C. Ré, "How to train your hippo: State space models with generalized orthogonal basis projections," *arXiv preprint arXiv:2206.12037*, 2022.

[35] A. Gu, I. Johnson, K. Goel, K. Saab, T. Dao, A. Rudra, and C. Ré, "Combining recurrent, convolutional, and continuous-time models with linear state space layers," *Advances in neural information processing systems*, vol. 34, pp. 572–585, 2021.

[36] A. Gu, T. Dao, S. Ermon, A. Rudra, and C. Ré, "Hippo: Recurrent memory with optimal polynomial projections," *Advances in neural information processing systems*, vol. 33, pp. 1474–1487, 2020.

[37] A. Tustin, "A method of analysing the behaviour of linear systems in terms of time series," *Journal of the Institution of Electrical Engineers-Part IIA: Automatic Regulators and Servo Mechanisms*, vol. 94, no. 1, pp. 130–142, 1947.

[38] A. Gu, K. Goel, A. Gupta, and C. Ré, "On the parameterization and initialization of diagonal state space models," *Advances in Neural Information Processing Systems*, vol. 35, pp. 35 971–35 983, 2022.

[39] D. Hendrycks and K. Gimpel, "Gaussian error linear units (gelus)," *arXiv preprint arXiv:1606.08415*, 2016.

[40] S. Chopra, R. Hadsell, and Y. LeCun, "Learning a similarity metric discriminatively, with application to face verification," in *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, vol. 1. IEEE, 2005, pp. 539–546.

[41] D. P. Kingma, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[42] I. Higgins, L. Matthey, A. Pal, C. P. Burgess, X. Glorot, M. M. Botvinick, S. Mohamed, and A. Lerchner, "beta-vae: Learning basic visual concepts with a constrained variational framework." *ICLR (Poster)*, vol. 3, 2017.

[43] L. Dinh, J. Sohl-Dickstein, and S. Bengio, "Density estimation using real nvp," *arXiv preprint arXiv:1605.08803*, 2016.

[44] I. Kobyzev, S. J. Prince, and M. A. Brubaker, "Normalizing flows: An introduction and review of current methods," *IEEE transactions on pattern analysis and machine intelligence*, vol. 43, no. 11, pp. 3964–3979, 2020.

[45] G. Papamakarios, E. Nalisnick, D. J. Rezende, S. Mohamed, and B. Lakshminarayanan, "Normalizing flows for probabilistic modeling and inference," *Journal of Machine Learning Research*, vol. 22, no. 57, pp. 1–64, 2021.

[46] D. P. Kingma and P. Dhariwal, "Glow: Generative flow with invertible 1x1 convolutions," *Advances in neural information processing systems*, vol. 31, 2018.

[47] J. Schulman, "Trust region policy optimization," *arXiv preprint arXiv:1502.05477*, 2015.

[48] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[49] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa *et al.*, "Isaac gym: High performance gpu-based physics simulation for robot learning," *arXiv preprint arXiv:2108.10470*, 2021.

[50] J. Fonollosa, E. Neftci, and M. Rabinovich, "Learning of chunking sequences in cognition and behavior," *PLoS computational biology*, vol. 11, no. 11, p. e1004592, 2015.

[51] L. Lai, A. Z. Huang, and S. J. Gershman, "Action chunking as policy compression," 2022.

[52] T. Z. Zhao, V. Kumar, S. Levine, and C. Finn, "Learning fine-grained bimanual manipulation with low-cost hardware," *arXiv preprint arXiv:2304.13705*, 2023.

Appendix A

Policy Configuration

Table A.1: Skill latent policy configuration.

Configuration	**Values**
Actor and Critic Networks	3-layer MLP with hidden dimensions: (512, 256, 128), ELU as activation.
Actor MLP layer initialization	orthogonal initialization with scaling of $\sqrt{2}$ (except last layer as 0.01) for weight and constant initialization for bias as 0.
Critic MLP layer initialization	orthogonal initialization with scaling of 1 for weight and constant initialization for bias as 0.
Learning algorithm	PPO
Initial action noise std	1.0
value loss coefficient	1.0
clipping range	0.2
Discount factor γ	0.99
GAE parameter λ	0.95
KL target	0.01
Entropy coefficient	0.0025

Table A.2: Refinement policy configuration.

Configuration	**Values**
Actor and Critic Networks	3-layer MLP with hidden dimensions: (32, 16, 16), ELU as activation.
Actor MLP layer initialization	orthogonal initialization with scaling of 1 for weight and constant initialization for bias as 0.
Critic MLP layer initialization	orthogonal initialization with scaling of 1 for weight and constant initialization for bias as 0.
Learning algorithm	PPO
Initial action noise std	1.0
value loss coefficient	1.0
clipping range	0.2
Discount factor γ	0.99
GAE parameter λ	0.95
KL target	1×10^{-5}
Entropy coefficient	0.001

Appendix B

Terrain Description

Terrain Name	Description	Type
Table Crouching	A terrain designed for crouching training to navigate under overhanging obstacles. Tables are placed with a minimum height of $0.45m$, covering 70% of the terrain, while the remaining 30% consists of sloped regions.	1
Gaps	A terrain designed for training jumping motions, where the robot jumps between boxes separated by gaps of up to $1.15m$.	1
Climb-up Box	A terrain designed for training the robot to climb up boxes with a maximum height of $0.8m$.	1
Stairs	A terrain designed for training the robot to climb up stairs with a maximum height of $0.2m$.	1
Obstacles	This terrain contains a mixture of high, randomly placed obstacles (80%) and sloped regions (20%).	2
Rock Piles	A challenging terrain consisting of rock piles of various sizes (approximately 80%) and sloped areas (20%), simulating a realistic rocky environment.	2
Simple Parkour	A terrain that combines stairs, gaps (up to $0.7m$), and boxes (up to $0.35m$) arranged to mimic a simple parkour-style environment without turns.	2

Table B.2: Description of testing terrains and their respective types.

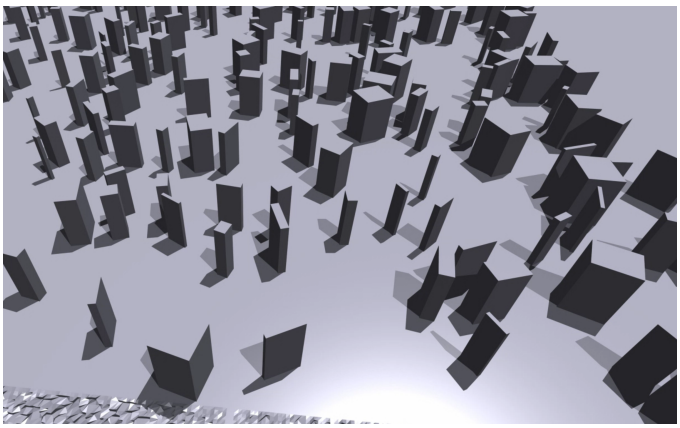


Figure B.1: Terrain: Obstacles.

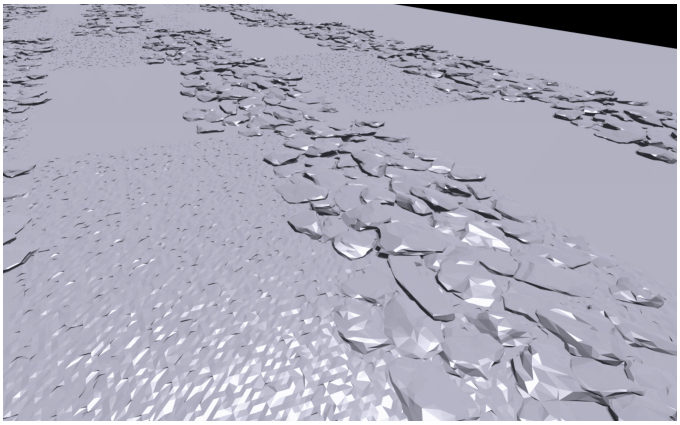


Figure B.2: Terrain: Rock Piles.

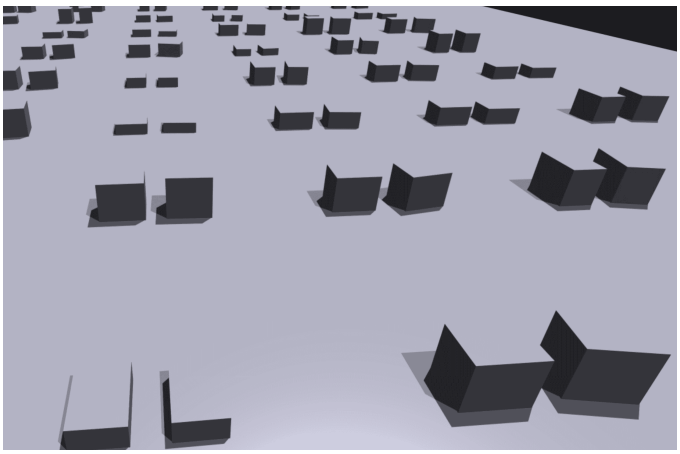


Figure B.3: Terrain: Gaps.

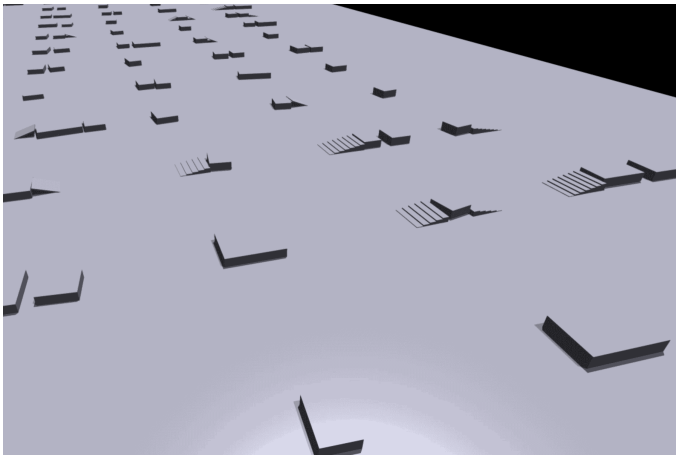


Figure B.4: Terrain: Simple Parkour.

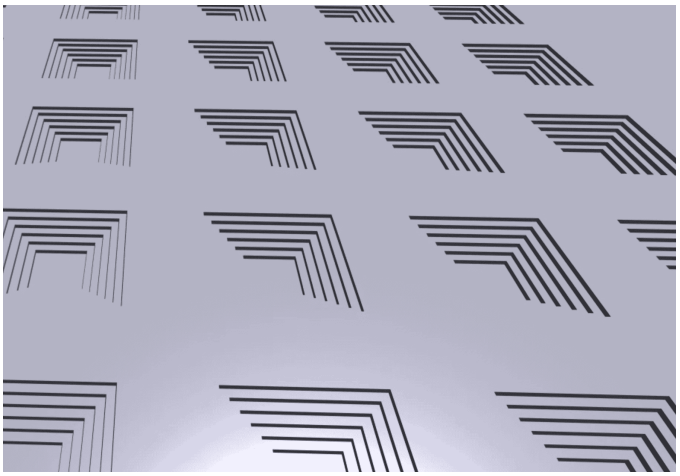


Figure B.5: Terrain: Stairs.