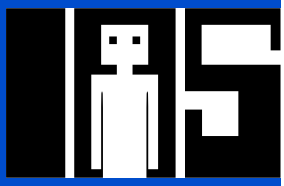
Learning Robot Locomotion from Diverse Datasets

Lernen der Roboterlokomotion aus vielfältigen Datensätzen
Master thesis by Lu Liu
Date of submission: February 18, 2025

1. Review: M.Sc. Michael Drolet
2. Review: Dr. Oleg Arenz
3. Review: Prof. Dr. Jan Peters
Darmstadt

TECHNISCHE
UNIVERSITÄT
DARMSTADT



Erklärung zur Abschlussarbeit gemäß § 22 Abs. 7 APB TU Darmstadt

Hiermit erkläre ich, Lu Liu, dass ich die vorliegende Arbeit gemäß § 22 Abs. 7 APB der TU Darmstadt selbstständig, ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt habe. Ich habe mit Ausnahme der zitierten Literatur und anderer in der Arbeit genannter Quellen keine fremden Hilfsmittel benutzt. Die von mir bei der Anfertigung dieser wissenschaftlichen Arbeit wörtlich oder inhaltlich benutzte Literatur und alle anderen Quellen habe ich im Text deutlich gekennzeichnet und gesondert aufgeführt. Dies gilt auch für Quellen oder Hilfsmittel aus dem Internet.

Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Falle eines Plagiats (§ 38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Darmstadt, 18. Februar 2025

L. Liu

Abstract

Quadrupedal robots have significant potential in applications such as search and rescue, exploration, and dynamic locomotion. While diverse motion datasets from animals and robotic platforms offer opportunities to improve motion quality and diversity, effectively learning locomotion from these datasets remains challenging. In this work, we address two key challenges: motion retargeting and the synthesis of motion sequences with flexible length.

For motion retargeting, we transfer motion sequences from sources with different sizes and morphologies (e.g., horse or robot Solo8) to a target quadrupedal robot (e.g., Unitree Go2). We adopt a spatial retargeting approach to ensure kinematic feasibility by maintaining consistent contact phases and minimizing foot sliding. For motion synthesis, we leverage existing frameworks, including a Vector Quantized Variational Autoencoder (VQ-VAE) and autoregressive models like Transformers, to encode, reconstruct, and generate diverse motion sequences of flexible length. These methods enable compact representations while preserving temporal coherence and motion structure. For motion imitation, we use a reinforcement learning approach inspired by DeepMimic, which employs time-step-based tracking rewards for efficient imitation.

We evaluate our framework through extensive experiments and ablation studies, leveraging a diverse dataset of dog, horse, and motion data from other robot platforms. Our results demonstrate successful motion retargeting across platforms (Go2 and A1) and highlight the potential for generalization. The generated dataset exhibits natural and diverse gaits, serving as a motion prior to guide the low-level policy in imitating reference trajectories from the high-level policy.

Zusammenfassung

Vierbeinige Roboter haben großes Potenzial in Anwendungen wie Such- und Rettungsmissionen, Exploration und dynamischer Fortbewegung. Während verschiedene Bewegungsdatensätze von Tieren und Robotern die Möglichkeit bieten, die Bewegungsqualität und -vielfalt zu verbessern, bleibt das effektive Erlernen der Fortbewegung aus diesen Datensätzen eine Herausforderung. In dieser Arbeit gehen wir zwei zentrale Herausforderungen an: Bewegungsretargeting und die Synthese von Bewegungssequenzen mit flexibler Länge.

Für das Bewegungsretargeting übertragen wir Bewegungssequenzen von Quellen mit unterschiedlichen Größen und Morphologien (z. B. Pferd oder Roboter Solo8) auf einen Zielroboter mit vier Beinen (z. B. Unitree Go2). Wir verwenden einen räumlichen Retargeting-Ansatz, um die kinematische Machbarkeit sicherzustellen, indem wir konsistente Kontaktphasen beibehalten und das Rutschen der Füße minimieren. Für die Bewegungssynthese nutzen wir bestehende Frameworks, einschließlich eines Vector Quantized Variational Autoencoders (VQ-VAE) und autoregressiver Modelle wie Transformer, um diverse Bewegungssequenzen flexibler Länge zu codieren, zu rekonstruieren und zu erzeugen. Diese Methoden ermöglichen kompakte Darstellungen, während sie die zeitliche Kohärenz und die Struktur der Bewegung bewahren. Für die Bewegungsimitation verwenden wir einen verstärkenden Lernansatz, inspiriert von DeepMimic, der auf zeitschrittbasierten Tracking-Belohnungen für eine effiziente Imitation setzt.

Wir evaluieren unser Framework durch umfangreiche Experimente und Ablationsstudien und nutzen einen vielfältigen Datensatz von Hund-, Pferd- und Bewegungsdaten anderer Roboterplattformen. Unsere Ergebnisse zeigen erfolgreiches Bewegungsretargeting über verschiedene Plattformen hinweg (Go2 und A1) und heben das Potenzial zur Generalisierung hervor. Der erzeugte Datensatz zeigt natürliche und vielfältige Gangarten und dient als Bewegungsprior, um die Low-Level-Policy bei der Imitation von Referenztrajektorien der High-Level-Policy zu unterstützen.

Contents

1. Introduction **2**

2. Foundations **4**
2.1. Inverse Kinematics 4
2.2. Vector Quantized Variational Autoencoder 5
2.3. Transformer 8
2.4. Reinforcement Learning 10
2.5. Sim-to-Real Transfer 12
2.6. Related Work 12

3. Methodology **15**
3.1. Unit Vector Method 15
3.2. Quantization-based Autoregressive Motion Synthesis 18
3.3. Proximal Policy Optimization with Imitation Objectives 23

4. Experimental Evaluation **25**
4.1. Metrics for Evaluating Motion Retargeting Performance 26
4.2. Motion Synthesis: From Reconstruction to Generation 28
4.3. Imitation Performance on Unitree Go2 32

5. Conclusion **37**

A. Appendix **44**

Figures and Tables

List of Figures

2.1. VAE Structure. The encoder maps the input **x** to a latent vector **z**, which is regularized by the prior distribution $p(\mathbf{z})$. The decoder reconstructs the input **x** from the latent vector **z**. 6

2.2. Left: A figure illustrating the VQ-VAE for image data **x** reconstruction. Right: A visualization of the embedding space. The output of the encoder $z_e(\mathbf{x})$ is mapped to the nearest point $\mathbf{e}_2$ from the embedding space. The gradient $\nabla_z L$ (in red) directs the encoder to modify its output, possibly affecting the parameters in the subsequent forward pass. 7

2.3. Left: Scaled Dot-Product Attention layer structure. Right: Multi-Head attention layer running in parallel 9

2.4. Interaction between agent and environment.The agent takes actions in the environment and receives rewards based on the actions taken. 10

3.1. Left: a common skeleton of the horse and dog. The blue points represent the withers and hips, while the red points represent the toes. Right: a common skeleton of the quadruped robot. The blue points represent the shoulder and hip joints, and the red points represent the feet. 17

3.2. Embedding Space: A single codebook is divided into three distinct codebooks. The final embedding vector is obtained by concatenating the codes from these individual codebooks. 20

3.3. VQ-VAE with Product Quantization. The input sequence $\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T\}$ is processed by the encoder, which includes convolutional layers and multi-head self-attention. The encoded continuous latent vector sequence $z_e(\tilde{\mathbf{X}})$ is then product-quantized into discrete codes using multiple codebooks. The quantized representations $z_q(\tilde{\mathbf{X}})$ are passed through a decoder, which mirrors the encoder's structure to reconstruct the full sequence $\{\hat{\mathbf{x}}_1, \hat{\mathbf{x}}_2, ..., \hat{\mathbf{x}}_T\}$ 21

3.4. Autoregressive Prediction: Without autoregressive head, the model predicts C targets in parallel. With autoregressive head, the model predicts C targets sequentially. 22

4.1. Results from Motion Retargeting: From top to bottom, the retargeted motions are dog pace, horse walk, mpc bound, and solo8 wave on Unitree GO2. The scaled original data are represented as nodes, providing a reference for the retargeted motions 28

4.2. Without Foot Regularizing (above) and With Foot Regularizing (below) . . 29

4.3. t-SNE Visualization of Latent Space with Four Codebooks 31

4.4. Reconstruction L2 Error by Gait and Motion Source: Comparison of models without EMA 33

4.5. Results of Gait Generation: From top to bottom, the generated motions are dog pace, horse walk, mpc bound, and solo8 wave. 34

4.6. Policy Learning Performance on Different Datasets 35

4.7. Training Performance with Domain Randomization and without Domain Randomization 36

A.1. Results from Motion Retargeting on Unitree A1: From top to bottom, the retargeted motions are dog trot, horse trot, mpc bound, and solo8 crawl . 44

A.2. t-SNE Visualization of Latent Space with One Codebook 45

A.3. t-SNE Visualization of the Latent Space Using a Single Codebook, Organized by Gaits. 46

A.4. t-SNE Visualization of the Latent Space Using Four Codebooks, Organized by Gaits. 47

A.5. t-SNE Visualization of the Latent Space Using a Single Codebook, Organized by Sources. 48
A.6. t-SNE Visualization of the Latent Space with Four Codebooks, Categorized by Data Sources: Dog, Horse, MPC, and Solo8. 49
A.7. Reconstruction L2 Error from model with EMA by Gait and Motion Source. 50
A.8. Visualization of codes in different codebooks: with EMA (above) and without EMA (below). 51

List of Tables

4.1. Gaits Available in Each Dataset. 26
4.2. Metrics for Evaluating Motion Retargeting Performance 27

A.1. Summary of Configuration Parameters for VQ-VAE and Generator 52
A.2. Summary of PPO Training Parameters 53
A.3. Penalty used in the reward function. 54
A.4. Observation Space.The height and trunk linear velocities are excluded from the observation space during actor training. 55
A.5. Domain Randomization Parameters. 56

1. Introduction

Quadrupedal robots have been an active research topic in robotics due to their potential applications in search and rescue, exploration, and dynamic locomotion. Recent advances in model-based controllers and reinforcement learning have enabled these robots to exhibit increasingly agile and adaptive behaviors. Moreover, diverse motion datasets from animals and robotic platforms offer opportunities to enhance the flexibility and adaptivity of quadrupedal motion. However, effectively learning robot locomotion from diverse datasets presents several challenges, which we address in this work.

The first challenge we address is motion retargeting, which involves transferring motion sequences from sources with different sizes and morphologies (e.g., an animal like a horse or another robot like Solo8) to the target quadrupedal robot like Unitree Go2. This task is nontrivial due to significant differences in kinematics, dynamics, and actuation constraints between the source and target systems. In our work, we adopt a spatial motion retargeting approach similar to [2][42], which ensures kinematic feasibility by maintaining consistent contact phases and minimizing foot sliding.

The second challenge lies in effectively representing motion sequences while preserving their underlying structure. Motion data is inherently high-dimensional with complex temporal dependencies, making it difficult to encode in a compact manner and reconstruct accurately. To address this, we adopt the framework from [25], which employs a Vector Quantized Variational Autoencoder (VQ-VAE) [27] to map motion sequences into discrete latent codes. Compared to traditional Variational Autoencoders (VAEs), VQ-VAE uses a discrete latent space instead of a continuous one. This design reduces the bit requirements for encoding. Additionally, VQ-VAE provides a one-to-one mapping between trajectory frames and latent codes. This design avoids the nondeterministic assignments of VAEs, which can cause the latent space to fail in capturing meaningful structures, a phenomenon known as posterior collapse. To capture temporal dependencies among tokens, the encoder and decoder leverage self-attention mechanisms [41] with causal masking. This design allows the model to focus on relevant temporal contexts while maintaining the sequential

nature of the data in the latent space. The codebook design enables the sampling of different codes that can be combined in various ways to generate diverse motion sequences. However, the diversity of the generated sequences is not solely determined by the VQ-VAE framework. It also depends on the richness of the training data and the size of the codebook. To further enhance diversity in latent space, the framework employs product quantization [16]. This design divides the latent representation into multiple subvectors, each of which is quantized into codes from different codebooks. Using this approach, the size of embedding space grows exponentially, enabling the decoder to produce higher-quality reconstructions with greater fidelity and detail. Moreover, for the sampling process, we adopt autoregressive models, such as Transformers [41], which make predictions of the next token via cross-entropy loss. This structure is well-suited for discrete sequences, By employing VQ-VAE, the dataset is discretized into a vocabulary of tokens, enabling seamless integration with the transformer framework. In contrast, VAEs generate continuous output, they cannot be naturally integrated into such frameworks, which limits their effectiveness for trajectory modeling.

To address motion imitation, where we adopt a reinforcement learning approach similar to DeepMimic [31], which employs a time-step-based tracking reward to imitate reference trajectories. An alternative method, such as Adversarial Motion Priors (AMP) [28], has been explored in prior work [7][43][42]. AMP utilizes a discriminator as a style reward, combining it with a task reward to accomplish various tasks while maintaining a specific motion style. We choose DeepMimic for its strong empirical performance in imitation tasks [2] [23], interpretability through explicit tracking rewards, higher sample efficiency without adversarial training, and seamless integration with our framework.

Finally, we present extensive experiments and ablation studies to evaluate our framework. We assess various network design choices. In contrast to other methods that rely on a single dog dataset, we retarget a diverse and extensive dataset for quadrupeds, including Matlab trot data [6], dog data [44], horse data [3], and Solo8 data [21]. This large, varied dataset enhances the generalization of our approach. The experiments are conducted on both the Go2 and A1 platforms, demonstrating a proof of concept that arbitrary platforms can be retargeted to arbitrary robots, as long as their kinematics are sufficiently similar. The quantization-based motion synthesis framework effectively reconstructs accurate motion datasets and generates a vast array of natural and diverse data for downstream low-level policy learning. We also apply domain randomization to the policy on a subset of our dataset, enabling potential transfer to real robots. However, due to time constraints, we were unable to evaluate the results on a real robot.

2. Foundations

In this chapter, we introduce the fundamental concepts and techniques utilized in this thesis. We begin by discussing the concept of inverse kinematics, which enables us to solve for the joint configurations, given desired task space positions. Next, we introduce the VQ-VAE and Transformer employed for data representation and generation, specifically motion sequences. Following that, we briefly introduce reinforcement learning, a technique used to learn the robot's control policy. Lastly, we cover sim-to-real transfer, which allows the transfer of policies learned in simulation to a real robot. Finally, we introduce the related work.

2.1. Inverse Kinematics

In robotics, forward kinematics is a technique used to compute the position of the end-effector, given its joint configuration. Mathematically, it can be expressed as $\mathbf{x}_e = f(\mathbf{q})$. where $\mathbf{q} \in \mathbb{R}^m$ represents the joint configuration, $\mathbf{x}_e \in \mathbb{R}^n$ denotes the position of the end effector, and f is the function that maps the joint configuration to the task space. This is a straightforward computation, compromised of a sequence of rigid body transformations, given the Denavit-Hartenberg parameters of the system.

Conversely, inverse kinematics involves determining the joint configuration of a robot based on the position of the end-effector. We will consider two popular approaches for robot control using inverse kinematics. The first approach involves using numerical optimization to solve for the joint configuration that satisfies a desired end effector pose. The second, often referred to as operational space control, uses the Jacobian pseudo-inverse method. We consider the latter approach for computational efficiency. Additionally, because we do not have any further constraints that could be optimized numerically (aside from matching

end-effector positions), this controller is suitable for our purposes. In general, we solve the following optimization problem:

$$\min_{\mathbf{q}} \|\mathbf{x}_e^* - f(\mathbf{q})\|_2 \,,$$

where $\mathbf{q}$ is the joint configuration and $\mathbf{x}_e^*$ represents the target position of the end-effector.

The Jacobian matrix $J(\mathbf{q}) \in \mathbb{R}^{n \times m}$ represents the linear mapping between joint velocities and end-effector velocities. The incremental changes in joint configuration can be calculated as follows:

$$\dot{\mathbf{q}}_t = J(\mathbf{q})^\dagger (\mathbf{x}_e^* - f(\mathbf{q}))$$

where $J(\mathbf{q})^\dagger = J(\mathbf{q})^T (J(\mathbf{q}) J(\mathbf{q})^T + \lambda \mathbf{I})^{-1}$ is the pseudo Jacobian, including a damping term λ. The updated joint configuration is then computed as:

$$\mathbf{q}_{t+1} = \mathbf{q}_t + \dot{\mathbf{q}}_t \cdot \Delta t \tag{2.1}$$

This process is repeated until the position of end-effector is sufficiently close to the desired position. There are many other established optimization methods, each with different constraints, to determine the changes in joint configuration. The details of these methods will not be discussed here, but will be covered in the methodology section.

2.2. Vector Quantized Variational Autoencoder

Generative models are a class of statistical models designed to learn the underlying distribution of data and generate new samples that resemble the original dataset. These models have gained significant attention due to their ability to create realistic data across various domains, including images, speech, video, and robotics. Among the most prominent generative models are Generative Adversarial Networks (GANs) [9], which employ a generator to produce samples and a discriminator to differentiate between real and generated data; Diffusion Models [14], which generate samples by progressively adding noise to data and learning to reverse this process; and Variational Autoencoders (VAEs) [18], which learn a latent representation of the data and optimize a lower bound on the data likelihood. In this work we focus on a particular adaptation of the VAEs, called the VQ-VAE.

VAEs typically consist of two main components: an encoder and a decoder, as illustrated in Figure 2.1.The input data $\mathbf{x}$ is first passed to the encoder, which compresses the information

from $\mathbf{x}$ into a latent space, represented as a latent vector $\mathbf{z}$. This latent vector $\mathbf{z}$ is then fed into the decoder, which reconstructs the information to produce an output $\hat{\mathbf{x}}$ that should closely resemble the original data $\mathbf{x}$. The goal of VAE is to maximize the evidence lower bound (ELBO) on the data likelihood, as defined below.

$$\mathcal{L}^{\text{ELBO}}_{\boldsymbol{\phi},\boldsymbol{\theta}} = \mathbb{E}_{\mathbf{x}}\mathbb{E}_{\mathbf{z}|\mathbf{x}}[\log p_{\boldsymbol{\phi}}(\mathbf{x} \mid \mathbf{z})] - \beta D_{\text{KL}}(q_{\boldsymbol{\theta}}(\mathbf{z} \mid \mathbf{x}) \| p(\mathbf{z}))$$

where $\mathbb{E}_{\mathbf{x}}\mathbb{E}_{\mathbf{z}|\mathbf{x}}[\log p_{\boldsymbol{\phi}}(\mathbf{x} \mid \mathbf{z})]$ serves as reconstruction loss, encouraging accurate data reconstruction, $D_{\text{KL}}(q_{\boldsymbol{\theta}}(\mathbf{z} \mid \mathbf{x}) \| p(\mathbf{z}))$ is the Kullback-Leibler divergence, which measures the distance between approximate posterior distribution $q_{\boldsymbol{\theta}}(\mathbf{z} \mid \mathbf{x})$ and the prior distribution $p(\mathbf{z})$. By minimizing the KL divergence, the latent space is effectively regularized, which prevents overfitting to the training data. The overall objective can be interpreted as a trade-off between two goals.

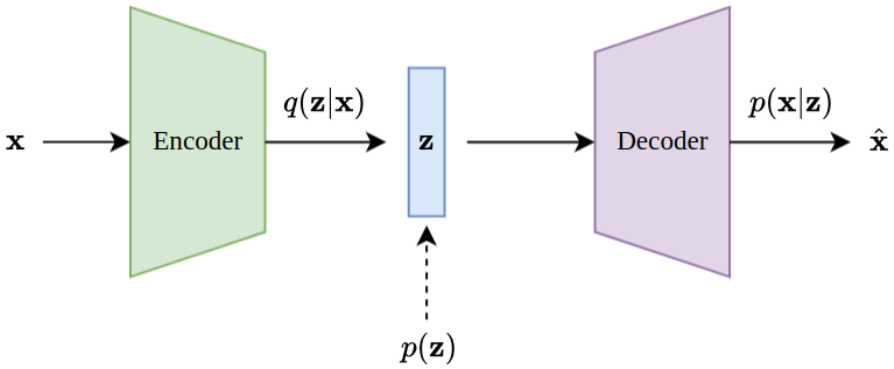


Figure 2.1.: VAE Structure. The encoder maps the input $\mathbf{x}$ to a latent vector $\mathbf{z}$, which is regularized by the prior distribution $p(\mathbf{z})$. The decoder reconstructs the input $\mathbf{x}$ from the latent vector $\mathbf{z}$.

VQ-VAE (Vector Quantized Variational Autoencoder) is a variant of the VAE. The key distinction between VQ-VAE and traditional VAEs lies in the representation of the latent

space: VQ-VAE deterministically maps an embedding to a quantized encoding. The discrete bootleneck design makes VQ-VAE more bit-efficient and scalable to high-dimensional datasets, unlike the standard VAEs.

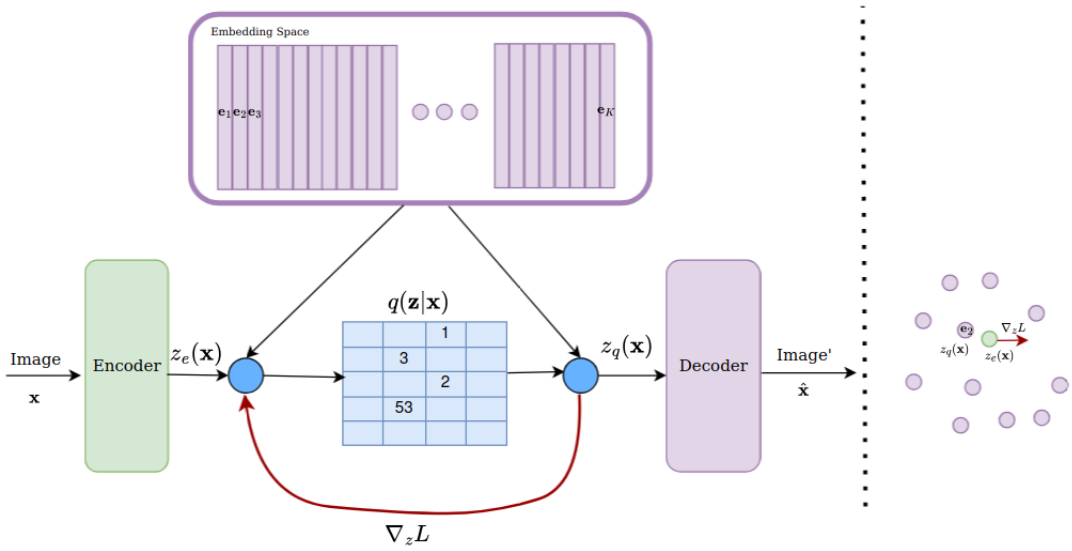


Figure 2.2.: Left: A figure illustrating the VQ-VAE for image data **x** reconstruction. Right: A visualization of the embedding space. The output of the encoder $z_e(\mathbf{x})$ is mapped to the nearest point $\mathbf{e}_2$ from the embedding space. The gradient $\nabla_z L$ (in red) directs the encoder to modify its output, possibly affecting the parameters in the subsequent forward pass.

In VQ-VAE, the latent variables are represented as a set of discrete code **e**, which collectively form a codebook. Each index i in the codebook points to a unique code $\mathbf{e}_i$. This discrete representation is achieved through vector quantization, a process that maps the continuous latent variables $z_e(\mathbf{x})$ to their closest discrete code $\mathbf{e}_k$ in the codebook. The decoder then reconstructs the input $\mathbf{x}$ from the quantized latent representation $z_q(\mathbf{x}) = \mathbf{e}_k$, where $k = \arg\min_i \|z_e(\mathbf{x}) - \mathbf{e}_i\|_2$. The entire process is depicted in Figure 2.2 right.

The training objective for VQ-VAE consists of the reconstruction loss, similar to that in VAE , commitment loss and codebook loss. The overall training loss is defined as follows:

$$L = \|\mathbf{x} - D(z_q(\mathbf{x}))\|_2^2 + \|\text{sg}[z_e(\mathbf{x})] - \mathbf{e}\|_2^2 + \beta\|z_e(\mathbf{x}) - \text{sg}[\mathbf{e}]\|_2^2 \tag{2.2}$$

The reconstruction loss $\|\mathbf{x} - D(z_q(\mathbf{x}))\|_2^2$ ensures that the decoded output $D(z_q(\mathbf{x}))$ closely matches the original input. Meanwhile the commitment loss $\|z_e(\mathbf{x}) - \text{sg}[\mathbf{e}]\|_2^2$ ensures that the encoder commits to an embedding. Similarly, the codebook loss $\|\text{sg}[z_e(\mathbf{x})] - \mathbf{e}\|_2^2$ moves the embedding table toward the encoder output. It is worth noting that the vector quantization process is non-differentiable, because of the argmin operation, which performs hard assignment. This non-differentiability poses challenges for gradient updates. To address this, VQ-VAE employs the argmin operation during the forward pass but bypasses it during the backward pass using the stop gradient operation. Specifically, the terms in loss function with stop gradient operation sg$[\cdot]$ during the gradient backpropagation will be treated as constant.

2.3. Transformer

The Transformer model was initially introduced for sequence-to-sequence tasks in natural language processing [41]. It has proven to be highly effective for various generative tasks [46, 25, 34, 45, 43, 37, 22].

The model typically follows an encoder-decoder structure, which is a popular architecture for sequence-to-sequence tasks. However, it is worth noting that the encoder is not always necessary. For certain tasks, for example, text or motion generation, a decoder-only model can be sufficient. In such cases, the model relies solely on the decoder, which is composed of multiple layers. Each layer typically includes a multi-head self-attention block and a feed-forward network, with residual connections and layer normalization applied between sub-layers. By chaining together multiple such layers, the decoder can effectively handle tasks without requiring an encoder.

The self-attention mechanism primarily calculates the attention relationships among tokens in the input sequence. Each token in the input is linearly mapped to three distinct embeddings, referred to as the query, key, and value. The attention scores are computed by taking the dot product of the query with all keys, dividing by the square root of the key's dimensionality d_k, and applying a softmax function to obtain normalized weights, as in Figure 2.3 left. These weights are then used to compute a weighted sum of the values. For example, given a sequence of input embeddings $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T)$, the self-attention

mechanism computes the attention scores as:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}})\mathbf{V}. \tag{2.3}$$

where query $\mathbf{Q} \in \mathbb{R}^{T \times d_k}$, key $\mathbf{K} \in \mathbb{R}^{T \times d_k}$, and value $\mathbf{V} \in \mathbb{R}^{T \times d_v}$.

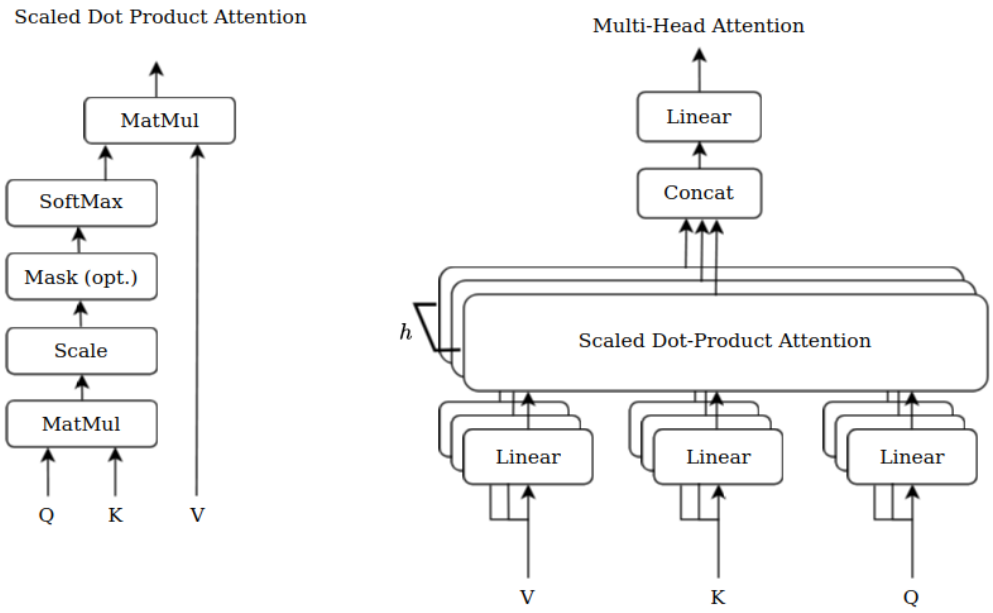


Figure 2.3.: Left: Scaled Dot-Product Attention layer structure. Right: Multi-Head attention layer running in parallel

The multi-head attention mechanism consists of multiple attention layers operating in parallel. Each layer independently generates an output, and these outputs are concatenated and then projected into the final values, as illustrated in Figure 2.3 right. It dynamically weights the importance of different positions in the input sequence, which is particularly important for in the context of motion synthesis. This capability allows the model to capture intricate temporal dependencies within the sequence, enabling it to generate effective action vocabulary indices.

2.4. Reinforcement Learning

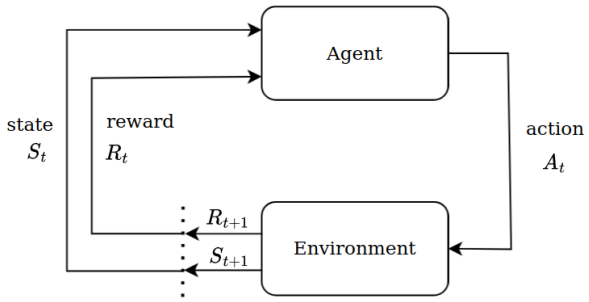


Figure 2.4.: Interaction between agent and environment.The agent takes actions in the environment and receives rewards based on the actions taken.

Reinforcement Learning (RL) is a machine learning technique that enables an agent to learn a policy by interacting with an environment.

The RL problem can be formulated as a Markov Decision Process (MDP), which is defined by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$. Here, $\mathcal{S}$ represents the state space, which contains all possible states the environment can be in. $\mathcal{A}$ denotes the action space, including all possible actions the agent can take. $\mathcal{P}$ is the transition probability function that describes all the probability of transitioning from one state to another, given an action. The reward function $\mathcal{R}$ computes the reward of being in a particular state and γ is the discount factor, which controls how much the agent should prioritize the future.

The entire interaction process is depicted in Figure 2.4. The agent observes the current state of the environment, denoted as state S_t, and receives a reward R_t. The agent then selects the action A_t, then gets a new state S_{t+1} from the environment and a new reward R_{t+1}. The agent's goal is to learn a policy π that can maximize the expected cumulative reward. The policy can be deterministic or stochastic and can be represented in various forms, such as lookup tables, decision trees, or neural networks. The choice of RL algorithm depends on the problem setup, particularly whether the state and action spaces are continuous or discrete. For instance, tabular RL uses tables to store state-action values and is ideal for small, discrete spaces. Q-learning, a value-based method, learns optimal policies by updating a Q-table. For continuous state spaces, its extension,

Deep Q-Networks (DQN), uses neural networks to approximate Q values and maximize expected future rewards. Alternatively, Proximal Policy Optimization (PPO) and other policy gradient methods are well suited for continuous spaces, balancing stability and efficiency by clipping policy updates to avoid large deviations. In this work, we employ policy gradient methods as our state and action spaces are both continuous.

In general, the expected cumulative reward can be mathematically stated as:

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\tau \sim \pi_{\boldsymbol{\theta}}} \left[\sum_{t=0}^{T} \gamma^t r_t \right]$$

where τ is the trajectory generated by the policy $\pi_{\boldsymbol{\theta}}$. γ is the discount factor. and r_t is the reward at time step t.

To maximize $J(\boldsymbol{\theta})$, we use gradient ascent:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$$

where α is the learning rate. The policy gradient theorem provides an expression for $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\tau \sim \pi_{\boldsymbol{\theta}}} \left[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_t \mid \mathbf{s}_t) A_t \right]$$

where A_t is the advantage function, which measures how much better an action $\mathbf{a}_t$ is compared to the average action at state $\mathbf{s}_t$.

To improve sample efficiency, we can use importance sampling to estimate the policy gradient using trajectories collected from an older policy $\pi_{\boldsymbol{\theta}_{\text{old}}}$:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\tau \sim \pi_{\boldsymbol{\theta}}} \left[\frac{\pi_{\boldsymbol{\theta}}(\mathbf{a}_t \mid \mathbf{s}_t)}{\pi_{\boldsymbol{\theta}_{\text{old}}}(\mathbf{a}_t \mid \mathbf{s}_t)} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\mathbf{a}_t \mid \mathbf{s}_t) A_t \right]$$

To stabilize training, we maximize a surrogate objective instead of directly maximizing $J(\boldsymbol{\theta})$:

$$L^{\text{CPI}}(\boldsymbol{\theta}) = \mathbb{E}_{\tau \sim \pi_{\boldsymbol{\theta}}} \left[\frac{\pi_{\boldsymbol{\theta}}(\mathbf{a}_t \mid \mathbf{s}_t)}{\pi_{\boldsymbol{\theta}_{\text{old}}}(\mathbf{a}_t \mid \mathbf{s}_t)} A_t \right]$$

where CPI stands for conservative policy iteration. However, this objective can lead to large policy updates, which can destabilize training. To prevent large updates, Proximal Policy Optimization (PPO) introduces a clipped surrogate objective:

$$L^{\text{CLIP}}(\boldsymbol{\theta}) = \mathbb{E}_{\tau \sim \pi_{\boldsymbol{\theta}}} \left[\min\left(\frac{\pi_{\boldsymbol{\theta}}(\mathbf{a}_t \mid \mathbf{s}_t)}{\pi_{\boldsymbol{\theta}_{\text{old}}}(\mathbf{a}_t \mid \mathbf{s}_t)} A_t, \text{clip}\left(\frac{\pi_{\boldsymbol{\theta}}(\mathbf{a}_t \mid \mathbf{s}_t)}{\pi_{\boldsymbol{\theta}_{\text{old}}}(\mathbf{a}_t \mid \mathbf{s}_t)}, 1-\epsilon, 1+\epsilon \right) A_t \right) \right]$$

where ϵ is a hyperparameter that controls the clipping range. The clipping ensures that the policy does not change too much in a single update, improving stability.

2.5. Sim-to-Real Transfer

Sim-to-Real transfer is a technique that enables the transfer of policies learned in simulation to a real robot. In policy gradient methods, there is a need to collect a large number of transitions, which is difficult in real physical systems but significantly easier and more cost-effective to achieve in simulation. However, simulation environments inevitably differ from reality. The goal of Sim-to-Real transfer is to bridge the gap between simulation and reality, ensuring that policies learned in simulation can be effectively applied to real-world scenarios.

Challenges in sim-to-real transfer often refer to the reality gap and the domain gap. The reality gap refers to the discrepancy between simulation and reality, such as the dynamics of the robot, the environment, and the sensors. The domain gap refers to the difference in the distribution of the data between simulation and reality, such as the lighting conditions, the textures, and the noise.

Domain randomization is one of the key techniques for sim-to-real transfer. The idea is to introduce random variations in the simulation so that the policy becomes robust enough to handle real-world variations. For example, in simulation MuJoCo, parameters like friction, object masses, joint dynamics, and sensor noise can be randomly sampled in a large range during training. The real-world environment and hardware can be seen as just one specific configuration within this range.

2.6. Related Work

Motion synthesis, which refers to the generation of realistic motion sequences, has been a prominent area of research. With the development of deep learning and generative models, significant progress has been made, particularly in computer graphics. Researchers have successfully employed generative models to synthesize human-like motion for animated characters [11][40][39]. There are two main approaches in motion synthesis: kinematics motion synthesis and physics-based motion synthesis.

Kinematics motion synthesis primarily relies on deep learning, machine learning, and data-driven methods. These approaches learn motion patterns from large datasets and generate new motion sequences without relying on physical simulations. The process is often end-to-end, where the corresponding motion sequence is generated given labels

such as action and duration. This method is widely used for animated characters in computer graphics, but the generated motions may not always adhere to physical laws. For instance, [8] combines Long Short-Term Memory (LSTM) networks with autoencoders to generate natural-looking human poses, leveraging the largest motion dataset at the time, [15]. Similarly, [1] employs an adapted Generative Adversarial Network (GAN) to predict future human poses based on previous frames, and validates their results on datasets [15] and [36]. [32] introduces a framework based on Variational Autoencoder (VAE) to generate realistic and diverse human motion sequences conditioned on actions, with the help of datasets [36] and [17]. Later, [25] adapts a new framework based on Vector-Quantized Variational Autoencoder (VQ-VAE) with a self-attention mechanism to synthesize natural human motion sequences based on actions and duration, utilizing multiple datasets, including BABEL [33], a newly introduced large-scale MoCap dataset at the time. Meanwhile, [22] leverages a Transformer-based architecture to generate diverse dance motions synchronized with music beats, using a massive collection of online dance videos. More recently, [39] utilized periodic autoencoder to synthesize smooth and natural motion transitions between two keyframes, supported by the dataset [13]. For some controllable characters, several works have explored combining reinforcement learning with latent representations. For example, [24] adopts the continuous latent space as the action space for downstream control algorithms in animated characters. In [20], the authors adapts [39] framework to further synthesize humanoid motion on MIT humanoid robot with the reatrgeted dataset from [31] and incorporates encoded motion segmentation information into the observation space, enhancing the performance of downstream motion learning algorithms.

Physics-based motion synthesis, on the other hand, employs forward dynamics and control policies such as model-based control or reinforcement learning to generate dynamically feasible motion sequences that adhere to the laws of physics. The motion representation is often controlled by policies. like in work [29], [31], [30], [47] [12] etc. our work is kinematics motion synthesis.

Quadrupedal locomotion learning has seen many advancements in recent years, with works such as [38] and [26] demonstrating the effectiveness of reinforcement learning in enabling agile and robust behaviors. However, pure reinforcement learning is prone to generate unnatural and stiff gaits. To achieve natural-looking gaits, various approaches have been explored. For instance, [2] builds on their prior work [31], applying reinforcement learning to train a robotic dog to successfully imitate reference animal gaits. Similarly, [7] leverages their previous framework [28] along with motion capture data from [44], enabling the Unitree A1 robot to perform natural gaits while meeting task requirements. More recent work, such as [12], introduces a vector-quantized discrete latent space to

capture the temopral and dynamic information within motion capture data and robot. Through multi-stage training, their method can be deployed on real robots, allowing them to execute strategic gameplay with highly natural locomotion.

Unlike their approach, our dataset includes a diverse range of gaits from animals [44, 3] and existing controllers across different robots [6, 21], providing a variety of gait options for learning in different scenarios. Our work does not integrate motion representation directly into the motion learning algorithm. Instead, motion synthesis is treated as a pre-trained process that generates reference trajectories solely based on labels while maintaining close similarity to the original data. These generated references then serve as motion priors for the motion learning problem, following a design similar to [2].

3. Methodology

In this chapter, we first introduce the motion retargeting algorithm, which is used to map motion data from diverse sources onto the target system. The motion data are sourced from animal motion capture datasets (e.g., dogs and horses) as well as motion records from existing controllers (e.g., a policy trained by reinforcement learning and model predictive controller) for various robot platforms. These datasets cover a wide range of gaits, such as trot, gallop, pace, and more. Using motion retargeting, we can convert these motion datasets into robot-compatible motion data. Next, we introduce the learning framework for motion representation. This framework employs vector quantization to discretize the latent space and, through product quantization, further divides the discrete latent vectors into multiple segments. This approach enlarges the latent space, enabling richer and more diverse motion representations. Then, we present the autoregressive prediction model using motion generation, which employs a multi-head self-attention mechanism with causal masking, to capture the distribution over the discrete latent action vocabulary indices, enabling diverse and coherent motion generation with flexible length. Finally, we discuss the motion imitation method, namely the Proximal Policy Optimization (PPO) algorithm with various imitation objectives.

3.1. Unit Vector Method

3.1.1. Datasets

The datasets used in this work can be divided into two main categories: motion capture datasets and robot motion datasets.

The motion capture datasets include data from dogs and horses. The dog motion capture dataset [44] contains various gaits such as walking, trotting, canter, and gallop. However, the motion clips for dogs are often lengthy and include many idle states. In this work,

we manually extract specific gait clips, such as trot, gallop, and pace, for further use. In contrast, the horse motion capture dataset [3] is well-structured, with each clip containing only a single gait (e.g., walk or trot).

The robot motion datasets consist of motion data from two sources: The Solo8 robot [10], a quadruped robot with an X-Type morphology [35], provides motion data generated by a trained PPO policy [21], including gaits such as walk, trot, crawl, bound, wave, and stilt. A model predictive controller (MPC) [6] provides motion data from a quadrupedal robot, including gaits such as gallop, trot, walk, pace, and bound.

However, the horse dataset is provided in relative coordinates. To analyze the walking velocity and trunk information, we first convert the data to absolute positions in the world frame. The contact points play a crucial role in this process. When a foot is in contact with the ground, its absolute position remains fixed. Based on this observation, we formulate the whole-body position estimation as the following optimization problem. We first set a threshold to determine whether a foot is in contact with the ground. The grounded feet contribute primarily to the body velocity. The velocity of the foot is then computed by the difference between the positions of the same foot in two consecutive frames. The problem can be formulated as below:

$$\min_{\mathbf{v}_{\text{base}}} \sum_{t=1}^{T} \sum_{i=1}^{4} \|(\mathbf{v}_{i,t} + \mathbf{v}_{\text{base},t})\|_2^2 \cdot c_{i,t} \quad \text{s.t.} \quad \|\mathbf{v}_{\text{base},t} - \mathbf{v}_{\text{base},t-1}\|_2^2 < \epsilon$$

where i denotes the foot index, t is the time index, T is the total time frames in one clip. $\mathbf{v}_{i,t}$ represents the linear velocity of the i-th foot at time t, $c_{i,t}$ is the binary contact indicator (e.g., 1 if the foot is in contact with the ground, 0 otherwise), $\mathbf{v}_{\text{base},t}$ is the body linear velocity at time t, and ϵ is the a threshold that ensures that the change in base velocity between consecutive time steps remains bounded. $\mathbf{v}_{base}$ is the body velocity in $\mathbb{R}^{3\times T}$.

3.1.2. Spatial Retargeting

The retargeting method we use is similar to [2] [42] [5]. It refers to the unit vector method [42]. Consider the keypoint position of the target system and source system are denoted as $\mathbf{p}^{\text{tar}}$, $\mathbf{p}^{\text{src}}$ respectively. In a given kinematic tree as in Figure 3.1, the directional vector between the key point j_{th} and its parent can be described as:

$$\mathbf{d}_j^{\text{src}} = \mathbf{p}_j^{\text{src}} - \mathbf{p}_{\mathcal{P}(j)}^{\text{src}}$$

where $\mathcal{P}(\cdot)$ represents the parent function in the kinematic tree. The j_{th} keypoint in the target system is then defined as:

$$\mathbf{p}_j^{\text{tar}} = \mathbf{p}_{\mathcal{P}(j)}^{\text{tar}} + \alpha \mathbf{d}_j^{\text{src}}$$

where α is the scaling factor, determined by the height ratio between the target system and the source system.

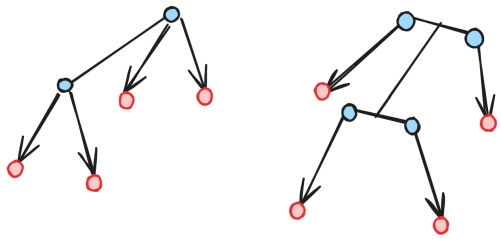


Figure 3.1.: Left: a common skeleton of the horse and dog. The blue points represent the withers and hips, while the red points represent the toes. Right: a common skeleton of the quadruped robot. The blue points represent the shoulder and hip joints, and the red points represent the feet.

We extract key points (e.g., withers, hips, toes) from horse and dog skeletons, as shown in Figure 3.1. We scaled the skeletons using the distance from the ground to the withers and map the kerpoints to corresponding points on the robot. The robot base position is the midpoint between the withers and the hips, and its orientation is derived from the hip to the withers vector. Foot positions are determined by directional vectors from withers or hips to toes.

Once the robot's foot positions are determined, we use differential inverse kinematics to compute the corresponding joint configurations. The problem can be formulated as follows:

$$\min_{\dot{\mathbf{q}}_t} (\mathbf{p}_t - \text{FK}(\mathbf{q}_t))^T (\mathbf{p}_t - \text{FK}(\mathbf{q}_t)) + (\dot{\mathbf{q}}_t - \dot{\mathbf{q}}_0)^T \mathbf{W} (\dot{\mathbf{q}}_t - \dot{\mathbf{q}}_0)$$

Here, t is the time index, $\text{FK}(\mathbf{q}_t)$ represents the forward kinematic process to get the current position of the feet, $\mathbf{p}_t$ is the target position of the feet, and $\mathbf{q}_t$ is the joint configuration. The term $\dot{\mathbf{q}}_0$ is defined as $\mathbf{K}(\mathbf{q}_{\text{default}} - \mathbf{q}_t)$, where $\mathbf{q}_{\text{default}}$ denotes the default joint configuration,

and **K** is a gain factor. In addition, **W** is the diagonal weight matrix. The solution to the above optimization problem is given by:

$$\dot{\mathbf{q}}_t = \mathbf{J}^{\dagger}(\mathbf{p}_t - \text{FK}(\mathbf{q}_t)) + (\mathbf{I} - \mathbf{J}^{\dagger}\mathbf{J})\dot{\mathbf{q}}_0$$

where $\mathbf{J}^{\dagger}$ is the pseudo-Jacobian matrix, $\mathbf{J}^{\dagger} = \mathbf{J}^T(\mathbf{J}\mathbf{J}^T + \lambda\mathbf{I})^{-1}$. λ is the damping factor to help avoid singularity in the inverse of matrix $\mathbf{J}\mathbf{J}^T$. Using Equation 2.1, we iteratively update $\mathbf{q}_t$ until the error in the task space converges to a sufficiently small value.

3.1.3. Foot Regularizing Constraint

We also consider the contact phase, as scaling all frames uniformly can distort it. For example, the horse is typically around 1.5 meters tall, whereas our robot is only 0.4 meters tall. This results in a scaling factor of approximately 0.3–0.4. After scaling, most of the toe positions tend to be much lower than before. Moreover, we lack ground truth data for the ground height to accurately determine the contact phase, as tracking points are usually mounted on the animal's ankle or other part of the foot, which are typically 1 to 2 cm above the ground. To maintain the consistency of the contact phase during spatial motion retargeting, we introduce the following constraint in the optimization process:

$$p_{i,t}^{\text{tar},z} = \begin{cases} p_{i,t}^{\text{src},z}, & \text{if } c_{i,t} = 1, \\ \beta p_{i,t}^{\text{src},z}, & \text{otherwise}, \end{cases}$$

where $p_{i,t}^{\text{src},z}$ is the original height position data from i-th foot at time step t, and β is the scaling factor. When the foot is in contact with the ground ($c_{i,t} = 1$), we retain the original height data. Otherwise, we use the scaled-down height.

3.2. Quantization-based Autoregressive Motion Synthesis

3.2.1. Self-Attention with Causal Mask

The autoregressive prediction mechanism is supported by a structure of self-attention with a causal mask. In this setup, the causal mask ensures that each token in the sequence can only attend to itself and the previous tokens, preventing information leakage from

future time steps. Given a sequence of input embeddings $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T)$, the attention equation from 2.3 can be rewrite as:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^T \cdot \mathbf{M}}{\sqrt{d_k}})\mathbf{V}.$$

where query $\mathbf{Q} \in \mathbb{R}^{T \times d_k}$, key $\mathbf{K} \in \mathbb{R}^{T \times d_k}$, and value $\mathbf{V} \in \mathbb{R}^{T \times d_v}$ are linear projections of the input sequence$x$. $\sqrt{d_k}$ is the scaling factor and $\mathbf{M}$ is the causal mask that assigns $-\infty$ to future positions, ensuring that they do not contribute to the calculation of attention.

3.2.2. Discrete Latent Space Representation

The latent space is structured as a discrete embedding table that contains K embedding vectors $\mathbf{e}_i \in \mathbb{R}^D$, where $i \in 1, 2, ..., K$. To further enhance the flexibility of the discrete representations learned by the encoder E, product quantization [16] is introduced. Each discrete latent vector $\mathbf{e}_i$ in the codebook is divided into C sections $(\mathbf{e}_i^1, \mathbf{e}_i^2, \ldots, \mathbf{e}_i^C) \in \mathbb{R}^{D/C}$, each belonging to different codebooks, as illustrated in Figure 3.2.

Given an input sequence $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T\}$, the sequence is first processed through a convolutional layer, reducing it to a compressed representation $\tilde{\mathbf{X}} = \{\tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_2, \ldots, \tilde{\mathbf{x}}_{T_d}\}$. This compressed sequence is then passed through multiple blocks, each consisting of a multi-head attention mechanism with positional encoding and a linear projection layer. This forms the encoder stage, where the output is denoted as $z_e(\tilde{\mathbf{X}}) = \{z_e(\tilde{\mathbf{x}}_1), z_e(\tilde{\mathbf{x}}_2), \ldots, z_e(\tilde{\mathbf{x}}_{T_d})\}$. Using self-attention with a causal mask, each encoded representation $z_e(\tilde{\mathbf{x}}_{t'})$ encapsulates information from previous frames, formally represented as:

$$q(z_e(\tilde{\mathbf{x}}_{t'}) \mid \tilde{\mathbf{x}}_{<t'}).$$

where t' denotes the timestep in compressed sequence $1, 2, ..., T_d$ after the convolutional layer, which differs from the original input sequence timesteps. Next, the encoded vector $z_e(\tilde{\mathbf{x}})$ undergoes product quantization, producing a discrete representation $z_q(\tilde{\mathbf{x}})$. Each element $z_q(\tilde{\mathbf{x}}_t)$ is assigned to a set of codebook indices:

$$z_q(\tilde{\mathbf{x}}_{t'}) = \mathcal{E}(i_{t'}^1, i_{t'}^2, \ldots, i_{t'}^C)$$

where $\mathcal{E}$ denotes the lookup function, extracting the corresponding codes from codebooks based on given indices $i \in [1, 2, \ldots, K]$ and $i_{t'}^c$ represents the index of the closest code in the c-th codebook at timestep t'. The decoder decodes the latent vectors then upsamples the compressed sequence using a convolutional layer again, reconstructing the entire

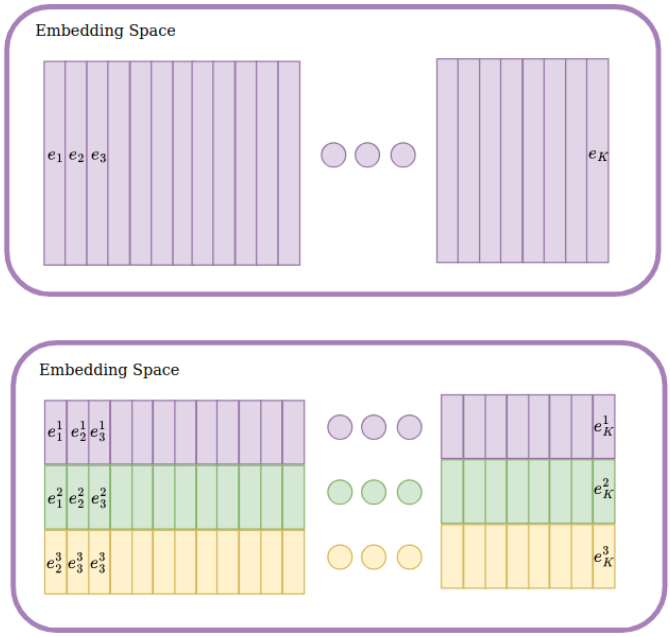


Figure 3.2.: Embedding Space: A single codebook is divided into three distinct codebooks. The final embedding vector is obtained by concatenating the codes from these individual codebooks.

sequence $\hat{\mathbf{X}} = \{\hat{\mathbf{x}}_1, \hat{\mathbf{x}}_2, \ldots, \hat{\mathbf{x}}_T\}$. The whole process is depicted in Figure 3.3. This strategy exponentially increases the size of the latent representaion to $K^{T_d \cdot C}$ possible combinations.

During training, a common issue occurs where some embedding vectors are updated frequently and drift away, while others remain stagnant and underutilized. Similarly to [25], we adopt an alternative codebook learning strategy based on exponential moving averages [19]. However, we find that this technique does not lead to significant performance improvements in our setup. Further details and analysis are discussed in the experimental

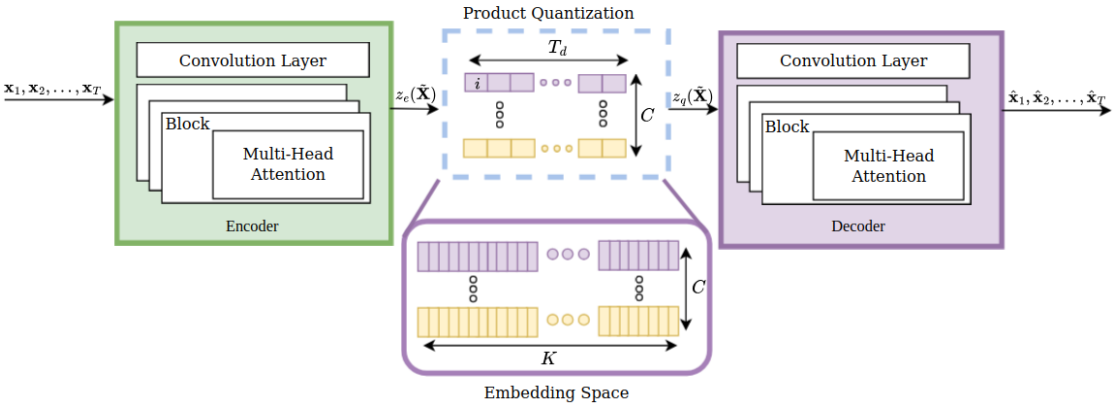


Figure 3.3.: VQ-VAE with Product Quantization. The input sequence $\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T\}$ is processed by the encoder, which includes convolutional layers and multi-head self-attention. The encoded continuous latent vector sequence $z_e(\tilde{\mathbf{X}})$ is then product-quantized into discrete codes using multiple codebooks. The quantized representations $z_q(\tilde{\mathbf{X}})$ are passed through a decoder, which mirrors the encoder's structure to reconstruct the full sequence $\{\hat{\mathbf{x}}_1, \hat{\mathbf{x}}_2, ..., \hat{\mathbf{x}}_T\}$

.

section. The updating process of the codes can be mathematically formulated as below:

$$N_i \leftarrow N_i \cdot \gamma + n_i(1-\gamma),$$
$$\mathbf{e}_i \leftarrow \frac{1}{N_i}(\mathbf{e}_i \cdot \gamma + \sum_{j}^{n_i} z_e(\tilde{\mathbf{x}}_j)(1-\gamma)).$$

where each codebook vector $\mathbf{e}_i$ is updated based on the previous one and the usage count N_i and the current closest usage count n_i. and γ denotes the discount factor.

Our framework builds on [25], but unlike the original work, which relies on extensive datasets from the Skinned Multi-Person Linear Model (SMPL)—a parametric humanoid model represented with meshes and requiring 80+ parameters to describe a pose—our dataset is joint-based, where only 12 parameters are sufficient to define a robot pose. This difference makes the original loss design incompatible with our data. To address this, we adjust the loss function, using a simple L2 loss. This adaptation successfully handles all

tasks, including high-quality reconstruction and generation, without needing to remap to a complex parametric space. This approach also extends the framework's potential application to humanoid robots and other robotic systems, as their data primarily consists of simple joint angles. Additionally, we employ training techniques such as min-max scaling and K-fold cross-validation.

3.2.3. Autoregressive Prediction over Discrete Latent Vector

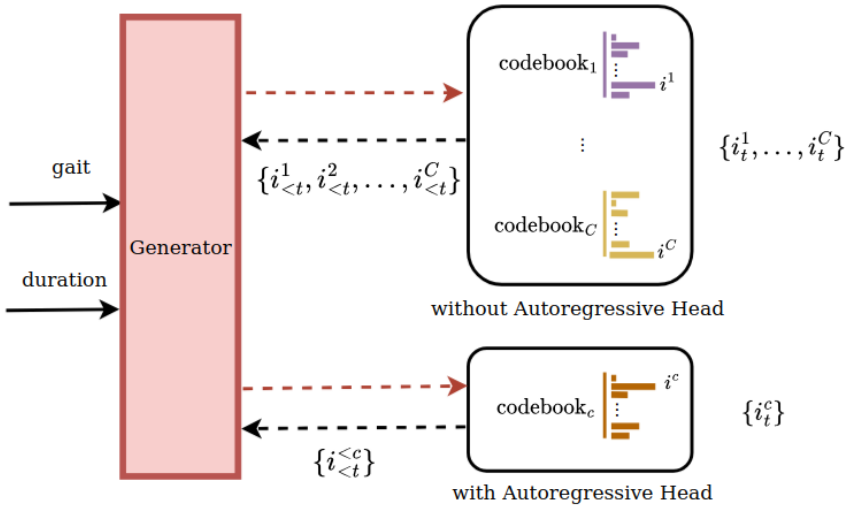


Figure 3.4.: Autoregressive Prediction: Without autoregressive head, the model predicts C targets in parallel. With autoregressive head, the model predicts C targets sequentially.

Based on the learned latent space, generating new motion sequences reduces to modeling the distribution over the action vocabulary indices. For example, consider a motion sequence of quadrupedal locomotion encoded into a set of indices $\mathbf{i}_{t'} \in \{i^1_{t'}, i^2_{t'}, ..., i^C_{t'}\}$ from different codebooks, where $t' \in 1, 2, ..., T_d$. The generator autoregressively predicts the indices using a self-attention mechanism with a causal mask, conditioned on a given gait label g and duration d. As the product quantization mechanism generates C targets, each focusing on different dimensions. Depending on the presence of an autoregressive

head [25], the generator can predict these indices either sequentially or in parallel, as illustrated in Figure 4.5.

Mathematically, the output of the generated sequence of indices probability is given by:

$$p(\mathbf{i}) = \prod_j p(\mathbf{i}_j \mid \mathbf{i}_{<j}, g, d)$$

The model is trained by maximizing the log-likelihood of this distribution. During the sampling stage, the indices are drawn sequentially from the distribution. The corresponding entries in the codebooks are then retrieved, passed through the decoder, and transformed into a full motion sequence.

3.3. Proximal Policy Optimization with Imitation Objectives

To guide the robot to follow the reference gait trajectories from the generator, we incorporate imitation objectives in reward function, following the work [31]. Let $\tau^g = \{\mathbf{g}_1, \mathbf{g}_2, ..., \mathbf{g}_T\}$ be a reference gait trajectory of the generator. where the $\mathbf{g}_t$ contains the goal state information for the quadrupedal robot at time step t, including the joint positions and velocities, the position and orientation of the trunk base, the linear velocities of the trunk base and the angular velocities. The reference trajectory is also incorporated into the observation space. For more details, please refer to Table A.4.

The joint position reward r_t^{jp} encourages the robot to minimize the difference between the joint positions of the target $\mathbf{q}^g$. This reward is formulated as an exponential function to penalize deviations from the desired joint configuration:

$$r_t^{\text{jp}} = \exp(-5\|\mathbf{q}_t^g - \mathbf{q}_t\|_2^2)$$

where $\mathbf{q}_t^g$ denotes the target position of joints at time step t. $\mathbf{q}_t$ represents the actual joint positions at time step t.

Similarly, the joint velocity reward r_t^{jv} is calculated according to the joint velocities:

$$r_t^{\text{jv}} = \exp(-0.1\|\dot{\mathbf{q}}_t^g - \dot{\mathbf{q}}_t\|_2^2)$$

The reward r_t^{toes} for the end-effectors, namely the toes:

$$r_t^{\text{toes}} = \exp(-40\|\mathbf{x}_t^{g,\text{toes}} - \mathbf{x}_t^{\text{toes}}\|_2^2)$$

where $\mathbf{x}_t^{\text{toes}}$ denotes the relative 3D position of toes with respect to the trunk base.

Finally, the trunk base position reward r_t^{tp} and trunk base velocity reward r_t^{tv} are fomulated as:

$$r_t^{\text{tp}} = \exp(-20\|\mathbf{x}_t^{g,\text{tp}} - \mathbf{x}_t^{\text{tp}}\|_2^2 - 10\|\mathbf{o}_t^{g,\text{tp}} - \mathbf{o}_t^{\text{tp}}\|_2^2)$$
$$r_t^{\text{tv}} = \exp(-2||\dot{\mathbf{x}}_t^{g,\text{tp}} - \dot{\mathbf{x}}_t^{\text{tp}}\|_2^2 - 0.2\|\dot{\mathbf{o}}_t^{g,\text{tp}} - \dot{\mathbf{o}}_t^{\text{tp}}\|_2^2)$$

Here, $\mathbf{x}_t^{g,\text{tp}}$ and $\dot{\mathbf{x}}_t^{g,\text{tp}}$ denote the goal global position and goal linear velocity of the robot's trunk base, respectively, while $\mathbf{o}_t^{g,\text{tp}}$ and $\dot{\mathbf{o}}_t^{g,\text{tp}}$ represent the goal orientation and goal angular velocity. Similarly, $\mathbf{x}_t^{\text{tp}}$ and $\dot{\mathbf{x}}_t^{\text{tp}}$ denote the actual global position and the actual linear velocity of the trunk base, and $\mathbf{o}_t^{\text{tp}}$ and $\dot{\mathbf{o}}_t^{\text{tp}}$ represent the actual orientation and actual angular velocity. Overall, the whole reward function is then defined as:

$$r_t = w^{\text{jp}} r_t^{\text{jp}} + w^{\text{jv}} r_t^{\text{jv}} + w^{\text{toes}} r_t^{\text{toes}} + w^{\text{tp}} r_t^{\text{tp}} + w^{\text{tv}} r_t^{\text{tv}}$$
$$w^{\text{jp}} = 0.5, w^{\text{jv}} = 0.05, w^{\text{toes}} = 0.2, w^{\text{tp}} = 0.15, w^{\text{tv}} = 0.1$$

The exponential form ensures that the reward decreases sharply as the deviation between the robot's state and the goal state increases, encouraging effective tracking of the reference trajectory. For more details on the training process, please refer to the Experimental Evaluation section.

4. Experimental Evaluation

In this chapter, we evaluate the motion retargeting algorithm on the Unitree Go2 robot. The Unitree Go2 has 12 joints in total, with 3 joints per leg: the hip, thigh, and calf. Each leg is equipped with 3 revolute actuators, which control the corresponding joints. For simulation, we use the MuJoCo environment. Additionally, we evaluated the motion reconstruction performance of the VQ-VAE model enhanced with product quantization. To optimize the model, we conduct a comprehensive hyperparameter search, exploring variables such as the number of codebooks, with exponential moving average technique or not.

Datasets Overview

We collect several datasets from publicly available sources to evaluate our motion retargeting algorithm. The datasets include:

- **Mocap Dog Dataset**: From [44], containing motions such as trot, gallop, and pace.
- **Mocap Horse Dataset**: From [3], including motions such as trotting and walking.
- **MPC-Controlled Quadrupedal Robot Dataset**: From [6], which we refer to as the **mpc** dataset. It contains motions such as trot, gallop, pace, bound, walk, and crawl.
- **Solo8 Robot Dataset**: From [21], controlled by an RL-based controller. We refer to this as the **solo8** dataset, which includes motions like trot, bound, walk, crawl, stilt, and wave.

Each combination of motion source and gait contains 10 trajectories, each approximately 2 seconds long, and all trajectories are resampled at a frequency of 50 Hz.

Motion Source	Gaits
dog	Trot, Gallop, Pace
horse	Trot, Walk
mpc	Trot, Gallop, Pace, Bound, Crawl
solo8	Trot, Bound, Walk, Crawl, Stilt, Wave

Table 4.1.: Gaits Available in Each Dataset.

4.1. Metrics for Evaluating Motion Retargeting Performance

To evaluate the performance of the motion retargeting algorithm, we use the following metrics.

Froude Number is a dimensionless parameter that characterizes dynamics similarity in legged locomotion, calculated as:

$$Fr = \frac{v^2}{gL}$$

where v is the velocity of the robot, g is the gravitational acceleration, and L is the leg length of the robot. According to previous research [4], two quadrupeds of different sizes exhibit similar dynamic movements when their Froude numbers are comparable. However, the Froude number alone is not a sufficient condition for dynamic similarity. It is included here as a reference indicator.

Joint limits error, which ensures that the retargetted motion adheres to the robot's physical constraints. The errors for joint position and velocity limits are calculated as:

$$e_{\text{Joint Pos}} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{T_i}\sum_{j=1}^{T_i}\sum_{k=1}^{K}\max(\left|q_{i,j,k} - q_k^{'}\right|, 0.0)$$

$$e_{\text{Joint Vel}} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{T_i}\sum_{j=1}^{T_i}\sum_{k=1}^{K}\max(\left|\dot{q}_{i,j,k} - \dot{q}_k^{'}\right|, 0.0)$$

where N is the total number of trajectories for a specific gait. and T_i is the length of i_{th} trajectory, K is the number of the joints. $q_{i,j,k}$ and $\dot{q}_{i,j,k}$ represent the joint positions and velocities, respectively, for the k-th joint at the time step j on the trajectory i, and $q_k^{'}$ and $\dot{q}_k^{'}$ are their corresponding joint limits.

Below is a table summarizing the metrics and their values for the evaluated gaits:

Gait	Froude Number		Violation	
	original	retarget	Joint Pos	Joint Vel
dog pace	0.260 ± 0.159	0.245 ± 0.050	0.0	0.000 ± 0.000
dog trot	0.781 ± 0.170	0.736 ± 0.160	0.0	0.001 ± 0.026
dog gallop	2.494 ± 1.160	2.350 ± 1.093	0.0	0.028 ± 0.434
horse walk	0.537 ± 0.237	0.133 ± 0.059	0.0	0.014 ± 0.247
horse trot	2.320 ± 0.159	0.580 ± 0.037	0.0	0.210 ± 1.059
mpc bound	0.106 ± 0.043	0.095 ± 0.038	0.0	0.036 ± 0.812
mpc trot	0.255 ± 0.081	0.227 ± 0.072	0.0	0.000 ± 0.000
mpc crawl	0.062 ± 0.029	0.055 ± 0.026	0.0	0.001 ± 0.014
mpc gallop	0.767 ± 0.280	0.685 ± 0.250	0.0	0.016 ± 0.173
mpc pace	0.131 ± 0.033	0.117 ± 0.029	0.0	2.980 ± 3.748
solo8 stilt	0.097 ± 0.085	0.071 ± 0.063	0.0	0.000 ± 0.000
solo8 bound	0.288 ± 0.178	0.213 ± 0.132	0.0	0.000 ± 0.000
solo8 wave	0.249 ± 0.055	0.184 ± 0.041	0.0	0.000 ± 0.009
solo8 crawl	0.096 ± 0.043	0.071 ± 0.032	0.0	0.000 ± 0.000
solo8 walk	0.101 ± 0.068	0.074 ± 0.051	0.0	0.000 ± 0.000
solo8 trot	0.106 ± 0.103	0.079 ± 0.076	0.0	0.000 ± 0.000

Table 4.2.: Metrics for Evaluating Motion Retargeting Performance

During retargeting, we apply foot regularizing constraints 3.1.3 to ensure the contact

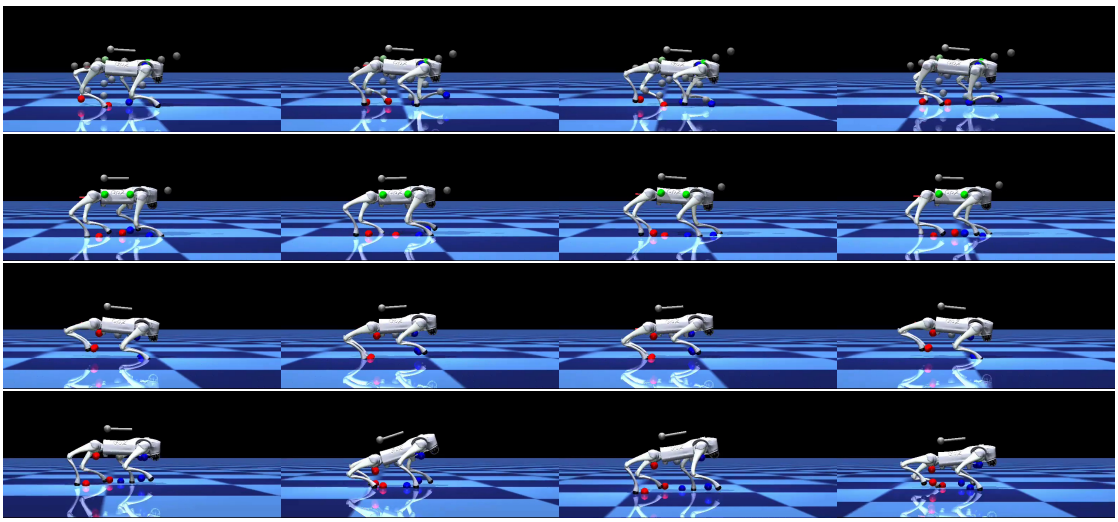


Figure 4.1.: Results from Motion Retargeting: From top to bottom, the retargeted motions are dog pace, horse walk, mpc bound, and solo8 wave on Unitree GO2. The scaled original data are represented as nodes, providing a reference for the retargeted motions

phases stay consistent with the reference motion. An example of this is shown in Figure 4.2. The ground truth is represented by the light color and dashed line, while the retargetted data is indicated by the solid block and line. With foot regularization, the contact phase is maintained. Additionally, due to heavy violations of joint velocity limits in the pace gait generated by the MPC, we excluded this gait from the evaluation in the motion representation and motion learning sections. The retargeting process is performed at the kinematic level, with simulation used for visualization. As shown in Figures 4.1 and A.1, the method successfully adapts the motion to different platforms and gaits.

4.2. Motion Synthesis: From Reconstruction to Generation

In this section, we assess the performance of the model in three key dimensions: latent space representation, motion reconstruction, and motion generation. We begin by demonstrating the effectiveness of our approach in clustering samples of the same gait and preserving the temporal structure of motion data, leading to high-quality reconstructions.

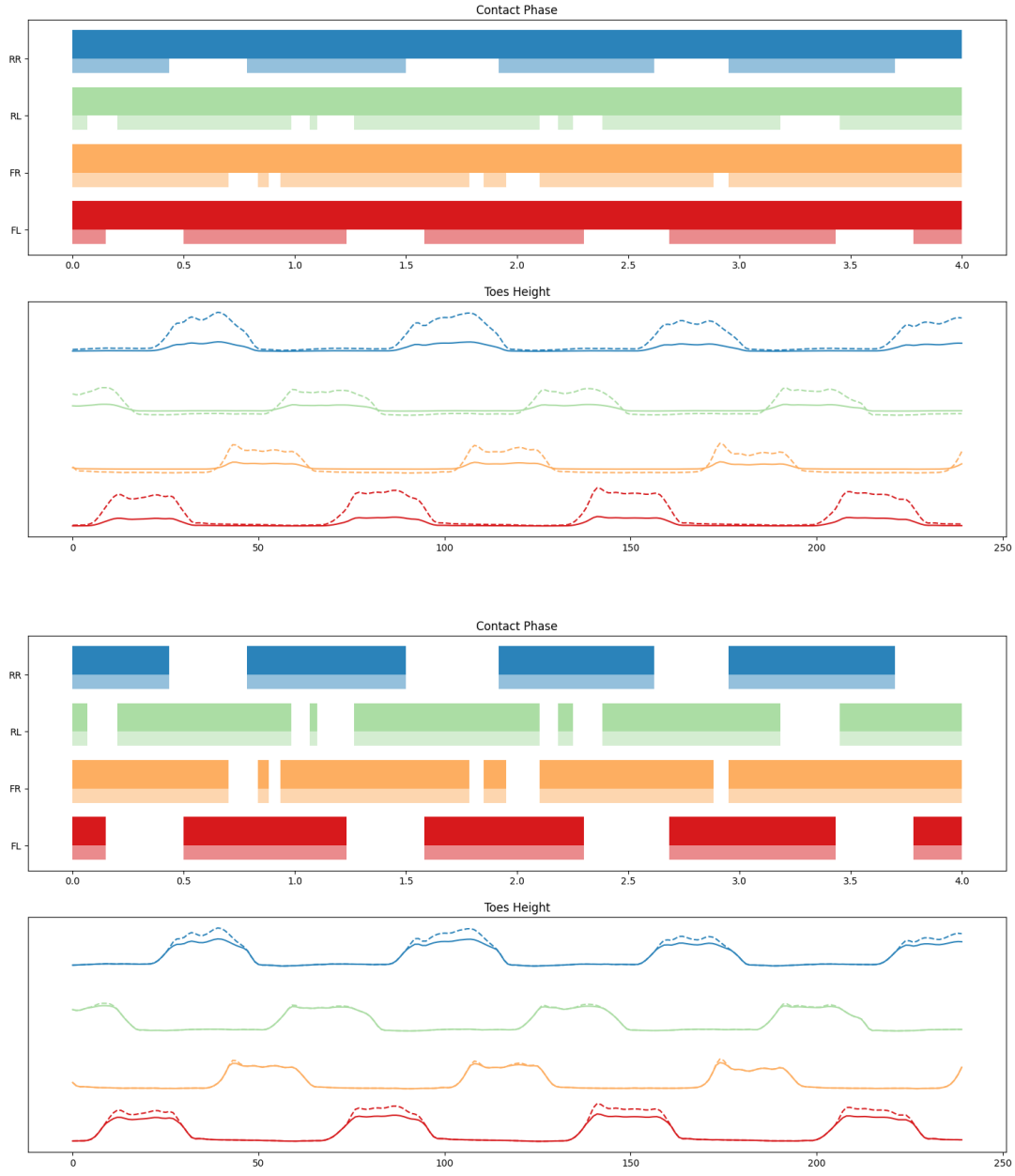


Figure 4.2.: Without Foot Regularizing (above) and With Foot Regularizing (below)

Next, we showcase the model's ability to produce realistic and diverse motion sequences.

4.2.1. Latent Space Representation

To visualize and analyze the structure of the learned latent space, we employ t-SNE (t-Distributed Stochastic Neighbor Embedding), a dimensionality reduction technique that projects high-dimensional data into a 2D or 3D space while preserving local relationships between points. Using t-SNE, we can easily analyze different design structures and their impact on latent space construction.

As shown in Figure 4.3, with four codebooks, the 15 different gaits are well clustered. In contrast, using a single codebook (Figure A.2 in the appendix) results in fewer distinct clustering. These results can be explained through the use of product quantization. By splitting the latent vector into subvectors and quantifying each separately, we effectively expand the latent space. A larger latent space captures hidden information within motion sequences, ultimately enabling more accurate motion reconstruction in the next step.

Additional diagrams are provided in the appendix, Figure A.3,A.4, A.5, A.6, where the latent codes are clustered by sources and gaits. This also demonstrates that using four codebooks leads to better clustering of various datasets in latent space. A visualization of the discrete latent space is illustrated in Figure A.8

4.2.2. Motion Reconstruction

For the motion reconstruction, our motion sequences dataset for training here includes trunk base velocity in the x-y plane while preserving the absolute z position, as the height varies across different gaits. For example, Solo8 tends to have a lower trunk position compared to most other gaits. Retaining this height information ensures that specific gait characteristics are preserved. Additionally, we include joint velocities and trunk base rotation, which is represented using the first two axes of the rotation matrix in a 6D format. To enhance reconstruction performance, we normalize the data to a range of [-1,1] using Min-Max scaling. We further divide the dataset into five subsets for K-fold cross-validation, which helps to improve the model accuracy in motion reconstruction.

The reconstruction error is measured using the L2 norm and is illustrated in Figure 4.4. Violin plots are used because they provide a more comprehensive view of the data distribution, capturing not only the mean, median, and quartiles, but also the spread and

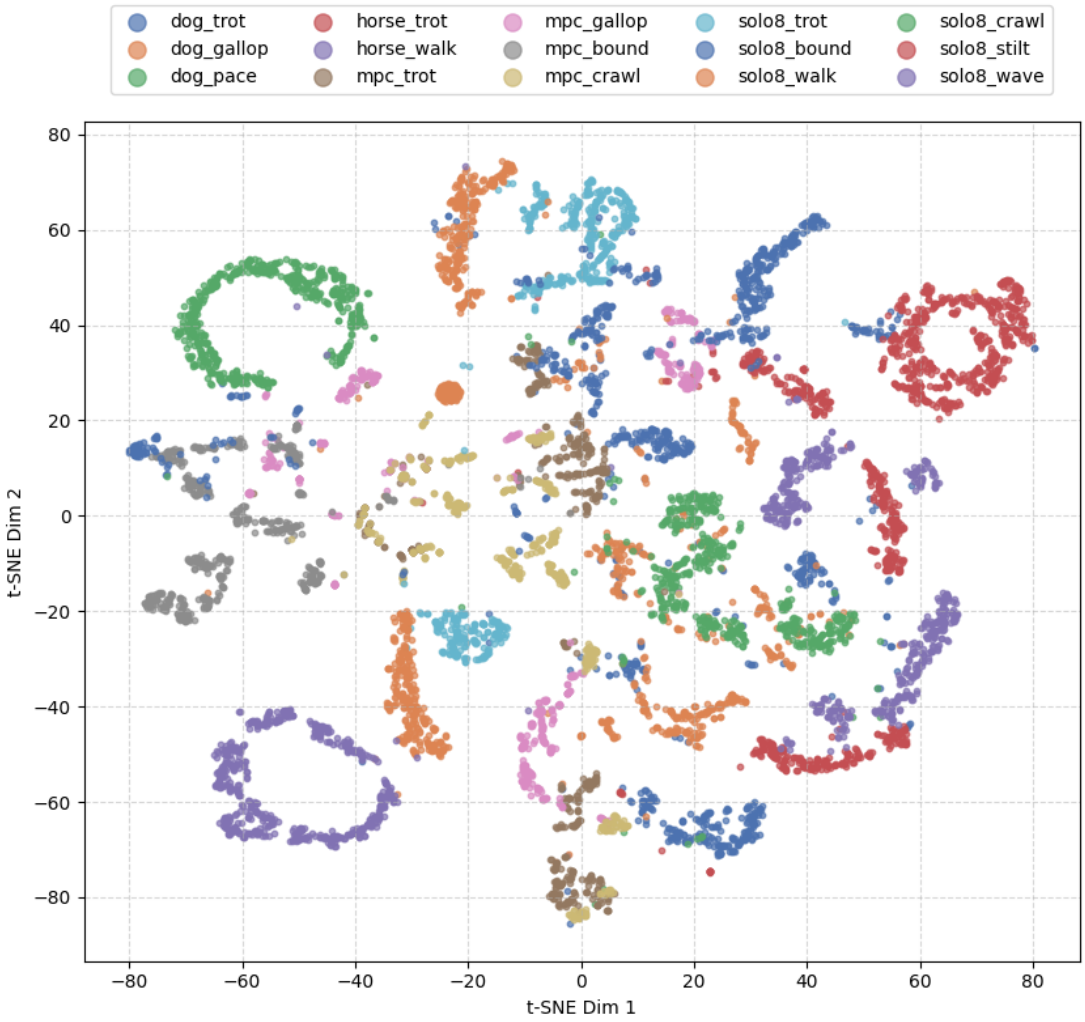


Figure 4.3.: t-SNE Visualization of Latent Space with Four Codebooks

density of the data, including potential outliers. This makes them more informative than quartile-based summaries alone. For example, motion capture datasets from animals like dogs and horses exhibit greater variation in reconstruction error compared to MPC and Solo8 datasets. In the dog dataset, the reconstruction error for the gallop gait is noticeably larger and more variable than for other gaits. This is because the dog dataset contains a wider variety of gaits with greater diversity and natural variations. In contrast, the reconstruction error in the MPC and Solo8 gaits is more uniform, as they are designed manually or generated by controllers rather than being captured from real animals.

The original framework uses the exponential moving average (EMA) to stabilize the training. From the visualization of the codebook in A.8, it is evident that EMA effectively restricts all codes to a relatively small range of updates. However, in our experiments, training with EMA does not improve performance, as demonstrated in A.7. One potential reason for this could be that the datasets used to evaluate the method, which contains a large SMPL data set [15], covering more than a million pose samples, whereas our dataset is smaller. This difference may explain why EMA is not necessary to balance the training process in our context.

4.2.3. Motion Generation

In motion generation experiments, we demonstrate the ability of our model to synthesize realistic and diverse motion sequences from the learned latent space. By sampling from the discrete latent representation autoregressively, the generated motions such as dog trot and horse walk exhibit natural behaviors. Qualitative evaluations show that the generated motions with autoregressive head prediction are visually plausible and capture the essential characteristics of the original data, part of generated gaits is shown in Figure 4.5. The relevant training parameters are listed in Table A.1.

4.3. Imitation Performance on Unitree Go2

The low-level policy on Unitree Go2 is trained in the Mujoco, with training parameters detailed in Table A.2. The observation space is listed in Table A.4. The high-level policy generates reference trajectories, which the low-level policy is trained to track.

To accommodate different gaits, we use two one-hot encoders: one for the motion source (e.g., reference trajectories from horses, dogs, etc.) and another for the gait type (e.g., trot,

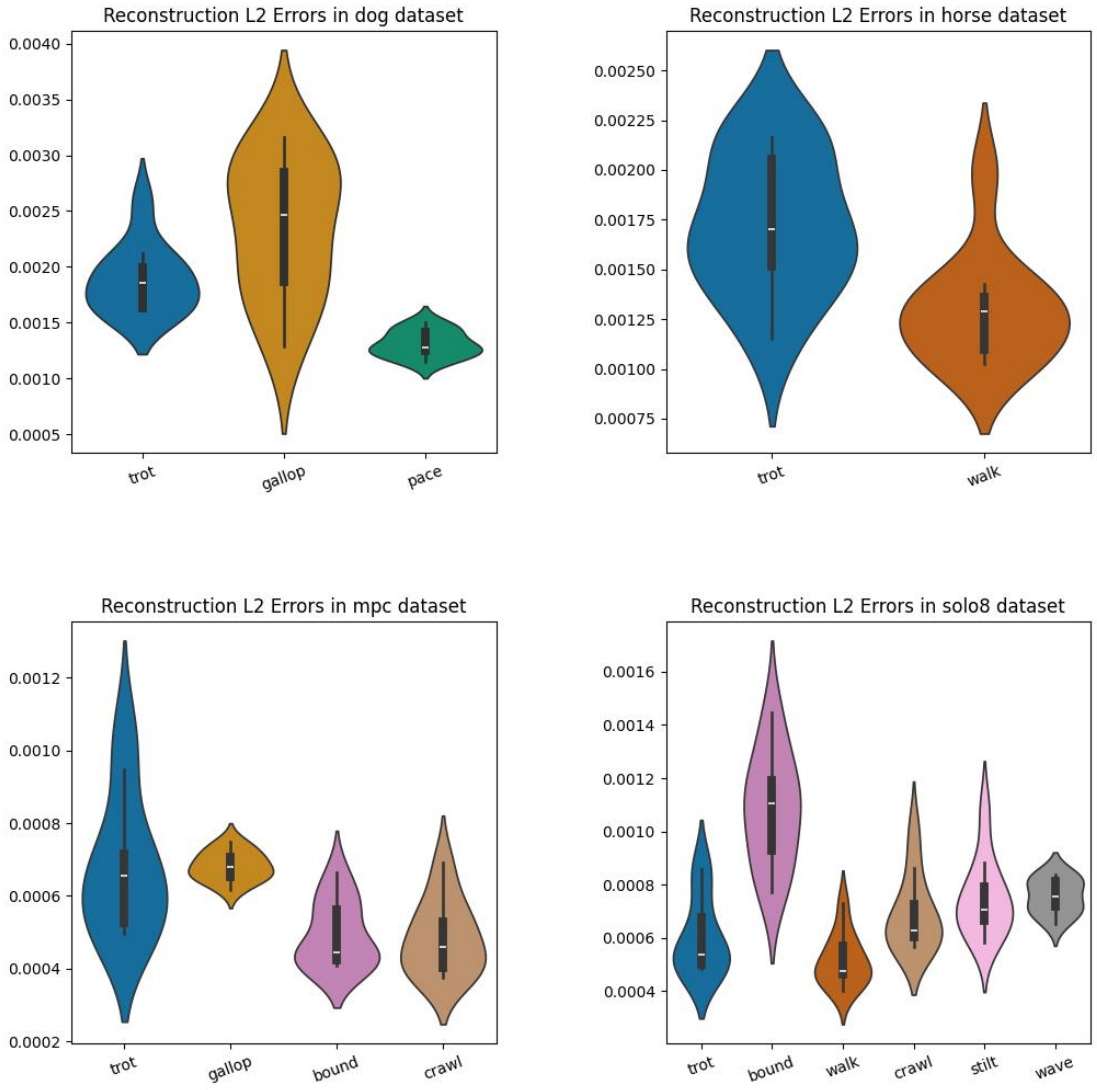


Figure 4.4.: Reconstruction L2 Error by Gait and Motion Source: Comparison of models without EMA

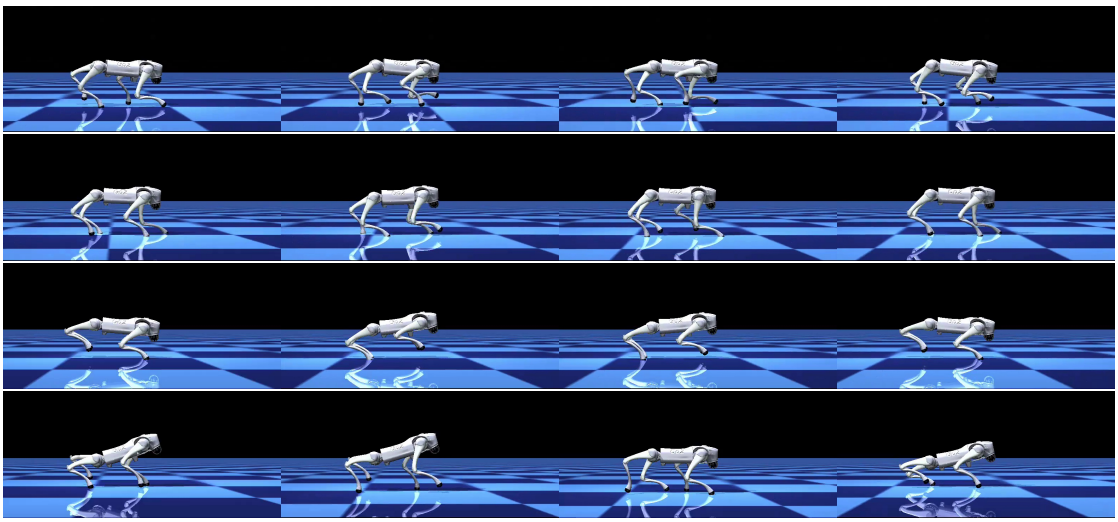


Figure 4.5.: Results of Gait Generation: From top to bottom, the generated motions are dog pace, horse walk, mpc bound, and solo8 wave.

pace, walk, etc.). We construct a buffer including 360 generated reference trajectories from 15 distinct gaits, each lasting approximately 2 seconds. At the start of each episode, a reference trajectory is randomly sampled from the buffer and integrated into the reward function defined in Chapter 3.3. The penalty reward terms are detailed in Table A.3.

Without domain randomization, training performance begins to converge after approximately 80 million steps. The agent successfully reproduces all buffer gaits and generalizes to newly generated gaits from the high-level policy, as shown in Figure 4.6. When domain randomization is introduced (with parameters listed in Table A.5), the agent is trained on a subset of gaits, such as trot and wave. A comparison of the results is presented in Figure 4.7.

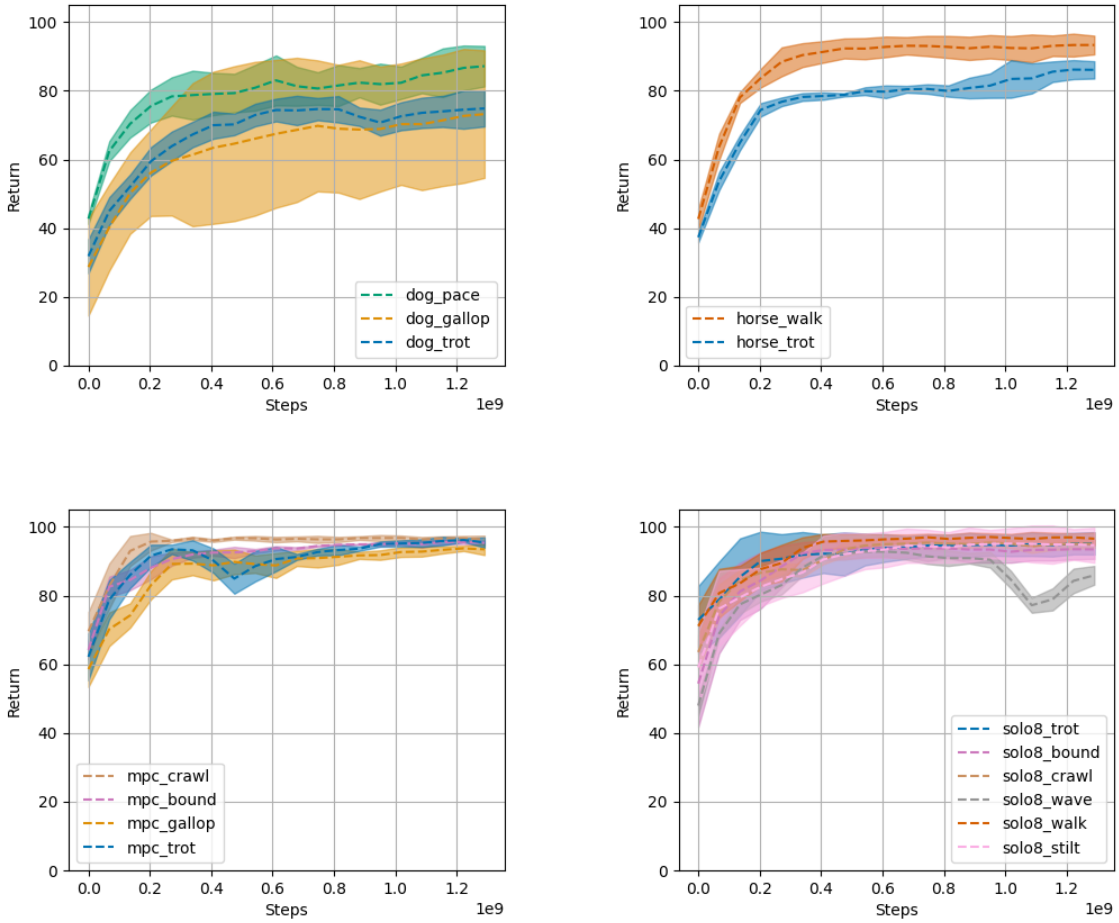


Figure 4.6.: Policy Learning Performance on Different Datasets

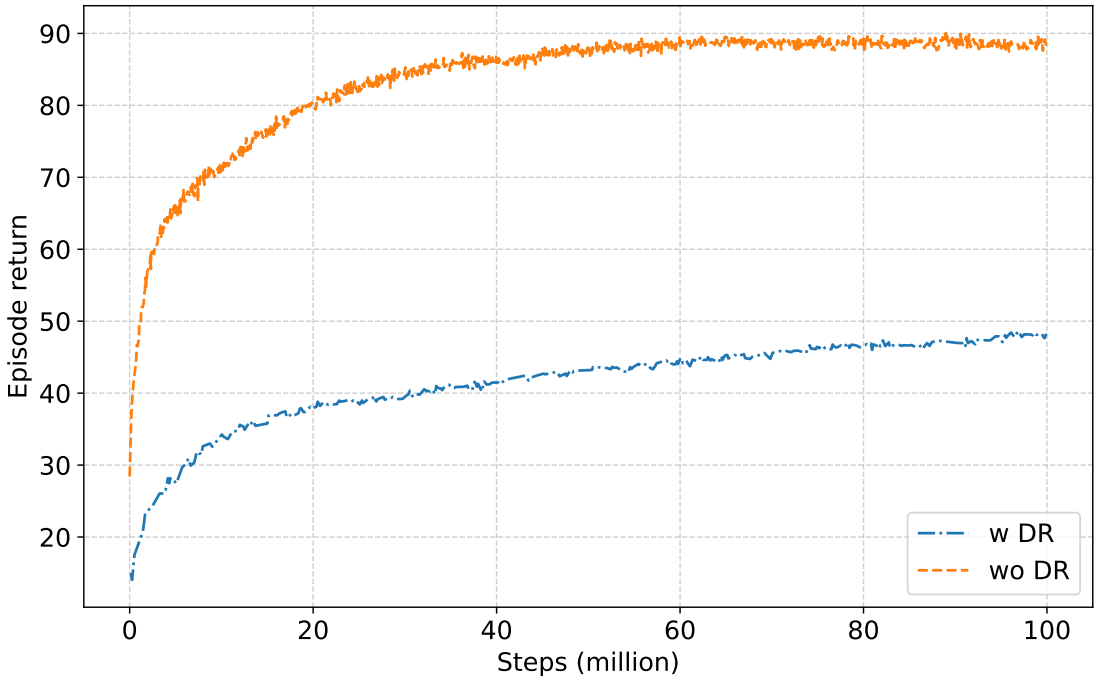


Figure 4.7.: Training Performance with Domain Randomization and without Domain Randomization

5. Conclusion

In this work, we presented a framework for motion retargeting, synthesis, and control that builds on existing methods. We demonstrated the effectiveness of our retargeting pipeline by successfully transferring motions from diverse sources—including various animal gaits and other robotic systems—onto a quadrupedal robot. These retargeted motion sequences serve as a structured motion dataset. Building upon existing frameworks, we encode the dataset into a discrete latent space. This discrete representation enables natural and diverse motion generation of arbitrary lengths by autoregressively sampling from learned action encoding indices. Unlike the original framework, we implement an alternative reconstruction loss to learn arbitrary (non-human-based) kinematics while still achieving high-quality motion generation (without the need for mapping into a higher-dimensional parametric space). The generated dataset is then used for downstream policy learning, where a single policy is trained to imitate 15 different gaits using a large set of generated trajectories as imitation objectives. The trained policy is able to imitate diverse gait behaviors while maintaining stability and natural motion. To help bridge the gap between simulation and reality, we incorporate domain randomization with the intention of deploying our work in real-world environments. Overall, our work advances robotic motion generation by providing an effective, scalable, and adaptable approach for motion retargeting and control across different robotic platforms.

Bibliography

[1] Emad Barsoum, John Kender, and Zicheng Liu. *HP-GAN: Probabilistic 3D human motion prediction via GAN*. Nov. 27, 2017. DOI: 10.48550/arXiv.1711.09561. arXiv: 1711.09561[cs]. URL: http://arxiv.org/abs/1711.09561.

[2] Xue Bin Peng et al. "Learning Agile Robotic Locomotion Skills by Imitating Animals". In: *Robotics: Science and Systems XVI*. Robotics: Science and Systems 2020. Robotics: Science and Systems Foundation, July 12, 2020. ISBN: 978-0-9923747-6-1. DOI: 10.15607/RSS.2020.XVI.064. URL: http://www.roboticsproceedings.org/rss16/p064.pdf.

[3] University of Bonn. *HORSE Project*. Accessed: 2024-02-09. 2024. URL: http://horse.cs.uni-bonn.de/index.html.

[4] Sharon R. Bullimore and J. Maxwell Donelan. "Criteria for dynamic similarity in bouncing gaits". In: *Journal of Theoretical Biology* 250.2 (Jan. 21, 2008), pp. 339–348. ISSN: 0022-5193. URL: https://www.sciencedirect.com/science/article/pii/S002251930700473O%7D.

[5] Sungjoon Choi and Joohyung Kim. "Towards a Natural Motion Generator: a Pipeline to Control a Humanoid based on Motion Data". In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). ISSN: 2153-0866. Nov. 2019, pp. 4373–4380. DOI: 10.1109/IROS40897.2019.8967941. URL: https://ieeexplore.ieee.org/document/8967941/?arnumber=8967941.

[6] Yanran Ding et al. "Representation-Free Model Predictive Control for Dynamic Motions in Quadrupeds". In: *IEEE Transactions on Robotics* 37.4 (Aug. 2021), pp. 1154–1171. ISSN: 1552-3098, 1941-0468. DOI: 10.1109/TRO.2020.3046415. arXiv: 2012.10002[cs]. URL: http://arxiv.org/abs/2012.10002.

[7] Alejandro Escontrela et al. *Adversarial Motion Priors Make Good Substitutes for Complex Reward Functions*. Mar. 28, 2022. DOI: 10.48550/arXiv.2203.15103. arXiv: 2203.15103[cs]. URL: http://arxiv.org/abs/2203.15103.

[8] Partha Ghosh et al. *Learning Human Motion Models for Long-term Predictions*. Dec. 3, 2017. DOI: 10.48550/arXiv.1704.02827. arXiv: 1704.02827[cs]. URL: http://arxiv.org/abs/1704.02827.

[9] Ian J. Goodfellow et al. *Generative Adversarial Networks*. June 10, 2014. DOI: 10.48550/arXiv.1406.2661. arXiv: 1406.2661[stat]. URL: http://arxiv.org/abs/1406.2661.

[10] Felix Grimminger et al. "An Open Torque-Controlled Modular Robot Architecture for Legged Locomotion Research". In: *IEEE Robotics and Automation Letters* 5.2 (Apr. 2020), pp. 3650–3657. ISSN: 2377-3766, 2377-3774. DOI: 10.1109/LRA.2020.2976639. arXiv: 1910.00093[cs]. URL: http://arxiv.org/abs/1910.00093.

[11] Chuan Guo et al. "Action2Motion: Conditioned Generation of 3D Human Motions". In: *Proceedings of the 28th ACM International Conference on Multimedia*. Oct. 12, 2020, pp. 2021–2029. DOI: 10.1145/3394171.3413635. arXiv: 2007.15240[cs]. URL: http://arxiv.org/abs/2007.15240.

[12] Lei Han et al. "Lifelike agility and play in quadrupedal robots using reinforcement learning and generative pre-trained models". In: *Nature Machine Intelligence* 6.7 (July 2024). Publisher: Nature Publishing Group, pp. 787–798. ISSN: 2522-5839. DOI: 10.1038/s42256-024-00861-3. URL: https://www.nature.com/articles/s42256-024-00861-3.

[13] Félix G. Harvey et al. "Robust Motion In-betweening". In: *ACM Transactions on Graphics* 39.4 (Aug. 31, 2020). ISSN: 0730-0301, 1557-7368. DOI: 10.1145/3386569.3392480. arXiv: 2102.04942[cs]. URL: http://arxiv.org/abs/2102.04942.

[14] Jonathan Ho, Ajay Jain, and Pieter Abbeel. *Denoising Diffusion Probabilistic Models*. Dec. 16, 2020. DOI: 10.48550/arXiv.2006.11239. arXiv: 2006.11239[cs]. URL: http://arxiv.org/abs/2006.11239.

[15] Catalin Ionescu et al. "Human3.6M: Large Scale Datasets and Predictive Methods for 3D Human Sensing in Natural Environments". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36.7 (July 2014). Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 1325–1339. ISSN: 1939-3539. DOI: 10.1109/TPAMI.2013.248. URL: https://ieeexplore.ieee.org/document/6682899/?arnumber=6682899.

[16] Herve Jégou, Matthijs Douze, and Cordelia Schmid. “Product Quantization for Nearest Neighbor Search”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33.1 (Jan. 2011). Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 117–128. ISSN: 1939-3539. DOI: 10.1109/TPAMI.2010.57. URL: https://ieeexplore.ieee.org/document/5432202/?arnumber=5432202.

[17] Yanli Ji et al. *A Large-scale Varying-view RGB-D Action Dataset for Arbitrary-view Human Action Recognition*. Apr. 24, 2019. DOI: 10.48550/arXiv.1904.10681. arXiv: 1904.10681[cs]. URL: http://arxiv.org/abs/1904.10681.

[18] Diederik P. Kingma and Max Welling. *Auto-Encoding Variational Bayes*. Dec. 10, 2022. DOI: 10.48550/arXiv.1312.6114. arXiv: 1312.6114[stat]. URL: http://arxiv.org/abs/1312.6114.

[19] Adrian Łańcucki et al. “Robust Training of Vector Quantized Bottleneck Models”. In: *2020 International Joint Conference on Neural Networks (IJCNN)*. July 2020, pp. 1–7. DOI: 10.1109/IJCNN48605.2020.9207145. arXiv: 2005.08520[cs]. URL: http://arxiv.org/abs/2005.08520.

[20] Chenhao Li et al. *FLD: Fourier Latent Dynamics for Structured Motion Representation and Learning*. Feb. 21, 2024. arXiv: 2402.13820[cs,eess]. URL: http://arxiv.org/abs/2402.13820.

[21] Chenhao Li et al. *Versatile Skill Control via Self-supervised Adversarial Imitation of Unlabeled Mixed Motions*. Feb. 11, 2023. DOI: 10.48550/arXiv.2209.07899. arXiv: 2209.07899[cs]. URL: http://arxiv.org/abs/2209.07899.

[22] Jiaman Li et al. *Learning to Generate Diverse Dance Motions with Transformer*. Aug. 18, 2020. DOI: 10.48550/arXiv.2008.08171. arXiv: 2008.08171[cs]. URL: http://arxiv.org/abs/2008.08171.

[23] Zhongyu Li et al. *Reinforcement Learning for Robust Parameterized Locomotion Control of Bipedal Robots*. Mar. 26, 2021. DOI: 10.48550/arXiv.2103.14295. arXiv: 2103.14295[cs]. URL: http://arxiv.org/abs/2103.14295.

[24] Hung Yu Ling et al. “Character Controllers Using Motion VAEs”. In: *ACM Transactions on Graphics* 39.4 (Aug. 31, 2020). ISSN: 0730-0301, 1557-7368. DOI: 10.1145/3386569.3392422. arXiv: 2103.14274[cs]. URL: http://arxiv.org/abs/2103.14274.

[25] Thomas Lucas et al. *PoseGPT: Quantization-based 3D Human Motion Generation and Forecasting*. Oct. 19, 2022. arXiv: 2210.10542[cs]. URL: http://arxiv.org/abs/2210.10542.

[26] Gabriel B. Margolis et al. *Rapid Locomotion via Reinforcement Learning*. May 5, 2022. DOI: 10.48550/arXiv.2205.02824. arXiv: 2205.02824[cs]. URL: http://arxiv.org/abs/2205.02824.

[27] Aaron van den Oord, Oriol Vinyals, and koray kavukcuoglu koray. "Neural Discrete Representation Learning". In: *Advances in Neural Information Processing Systems*. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/hash/7a98af17e63a0ac09ce2e96d03992fbc-Abstract.html.

[28] Xue Bin Peng et al. "AMP: Adversarial Motion Priors for Stylized Physics-Based Character Control". In: *ACM Transactions on Graphics* 40.4 (Aug. 31, 2021), pp. 1–20. ISSN: 0730-0301, 1557-7368. DOI: 10.1145/3450626.3459670. arXiv: 2104.02180[cs]. URL: http://arxiv.org/abs/2104.02180.

[29] Xue Bin Peng et al. "ASE: Large-Scale Reusable Adversarial Skill Embeddings for Physically Simulated Characters". In: *ACM Transactions on Graphics* 41.4 (July 2022), pp. 1–17. ISSN: 0730-0301, 1557-7368. DOI: 10.1145/3528223.3530110. arXiv: 2205.01906[cs]. URL: http://arxiv.org/abs/2205.01906.

[30] Xue Bin Peng et al. "DeepLoco: dynamic locomotion skills using hierarchical deep reinforcement learning". In: *ACM Transactions on Graphics* 36.4 (Aug. 31, 2017), pp. 1–13. ISSN: 0730-0301, 1557-7368. DOI: 10.1145/3072959.3073602. URL: https://dl.acm.org/doi/10.1145/3072959.3073602.

[31] Xue Bin Peng et al. "DeepMimic: example-guided deep reinforcement learning of physics-based character skills". In: *ACM Transactions on Graphics* 37.4 (Aug. 31, 2018), pp. 1–14. ISSN: 0730-0301, 1557-7368. DOI: 10.1145/3197517.3201311. URL: https://dl.acm.org/doi/10.1145/3197517.3201311.

[32] Mathis Petrovich, Michael J. Black, and Gül Varol. *Action-Conditioned 3D Human Motion Synthesis with Transformer VAE*. Sept. 19, 2021. DOI: 10.48550/arXiv.2104.05670. arXiv: 2104.05670[cs]. URL: http://arxiv.org/abs/2104.05670.

[33] Abhinanda R. Punnakkal et al. *BABEL: Bodies, Action and Behavior with English Labels*. June 23, 2021. DOI: 10.48550/arXiv.2106.09696. arXiv: 2106.09696[cs]. URL: http://arxiv.org/abs/2106.09696.

[34] Aditya Ramesh et al. *Zero-Shot Text-to-Image Generation*. Feb. 26, 2021. arXiv: 2102.12092[cs]. URL: http://arxiv.org/abs/2102.12092.

[35] Leanne Raw, Callen Fisher, and Amir Patel. “Effects of Limb Morphology on Transient Locomotion in Quadruped Robots”. In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). ISSN: 2153-0866. Nov. 2019, pp. 3349–3356. DOI: 10.1109/IROS40897.2019.8968206. URL: https://ieeexplore.ieee.org/document/8968206/?arnumber=8968206.

[36] Amir Shahroudy et al. *NTU RGB+D: A Large Scale Dataset for 3D Human Activity Analysis*. Apr. 11, 2016. DOI: 10.48550/arXiv.1604.02808. arXiv: 1604.02808[cs]. URL: http://arxiv.org/abs/1604.02808.

[37] Li Siyao et al. *Bailando: 3D Dance Generation by Actor-Critic GPT with Choreographic Memory*. Mar. 25, 2022. DOI: 10.48550/arXiv.2203.13055. arXiv: 2203.13055[cs]. URL: http://arxiv.org/abs/2203.13055.

[38] Laura Smith, Ilya Kostrikov, and Sergey Levine. *A Walk in the Park: Learning to Walk in 20 Minutes With Model-Free Reinforcement Learning*. Aug. 16, 2022. DOI: 10.48550/arXiv.2208.07860. arXiv: 2208.07860[cs]. URL: http://arxiv.org/abs/2208.07860.

[39] Paul Starke et al. “Motion In-Betweening with Phase Manifolds”. In: *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 6.3 (Aug. 16, 2023), pp. 1–17. ISSN: 2577-6193. DOI: 10.1145/3606921. URL: https://dl.acm.org/doi/10.1145/3606921.

[40] Sebastian Starke, Ian Mason, and Taku Komura. “DeepPhase: periodic autoencoders for learning motion phase manifolds”. In: *ACM Transactions on Graphics* 41.4 (July 2022), pp. 1–13. ISSN: 0730-0301, 1557-7368. DOI: 10.1145/3528223.3530178. URL: https://dl.acm.org/doi/10.1145/3528223.3530178.

[41] Ashish Vaswani et al. *Attention Is All You Need*. Aug. 1, 2023. arXiv: 1706.03762[cs]. URL: http://arxiv.org/abs/1706.03762.

[42] Taerim Yoon et al. *Spatio-Temporal Motion Retargeting for Quadruped Robots*. Apr. 17, 2024. arXiv: 2404.11557[cs]. URL: http://arxiv.org/abs/2404.11557.

[43] Fatemeh Zargarbashi et al. *RobotKeyframing: Learning Locomotion with High-Level Objectives via Mixture of Dense and Sparse Rewards*. July 16, 2024. arXiv: 2407.11562[cs]. URL: http://arxiv.org/abs/2407.11562.

[44] He Zhang et al. “Mode-adaptive neural networks for quadruped motion control”. In: *ACM Transactions on Graphics* 37.4 (Aug. 31, 2018), pp. 1–11. ISSN: 0730-0301, 1557-7368. DOI: 10.1145/3197517.3201366. URL: https://dl.acm.org/doi/10.1145/3197517.3201366.

[45] Zhanjie Zhang et al. *Rethink Arbitrary Style Transfer with Transformer and Contrastive Learning*. Apr. 21, 2024. arXiv: 2404.13584[cs]. URL: http://arxiv.org/abs/2404.13584.

[46] Haoyi Zhou et al. *Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting*. Mar. 28, 2021. arXiv: 2012.07436[cs]. URL: http://arxiv.org/abs/2012.07436.

[47] Qingxu Zhu et al. *Neural Categorical Priors for Physics-Based Character Control*. Oct. 6, 2023. arXiv: 2308.07200[cs]. URL: http://arxiv.org/abs/2308.07200.

A. Appendix

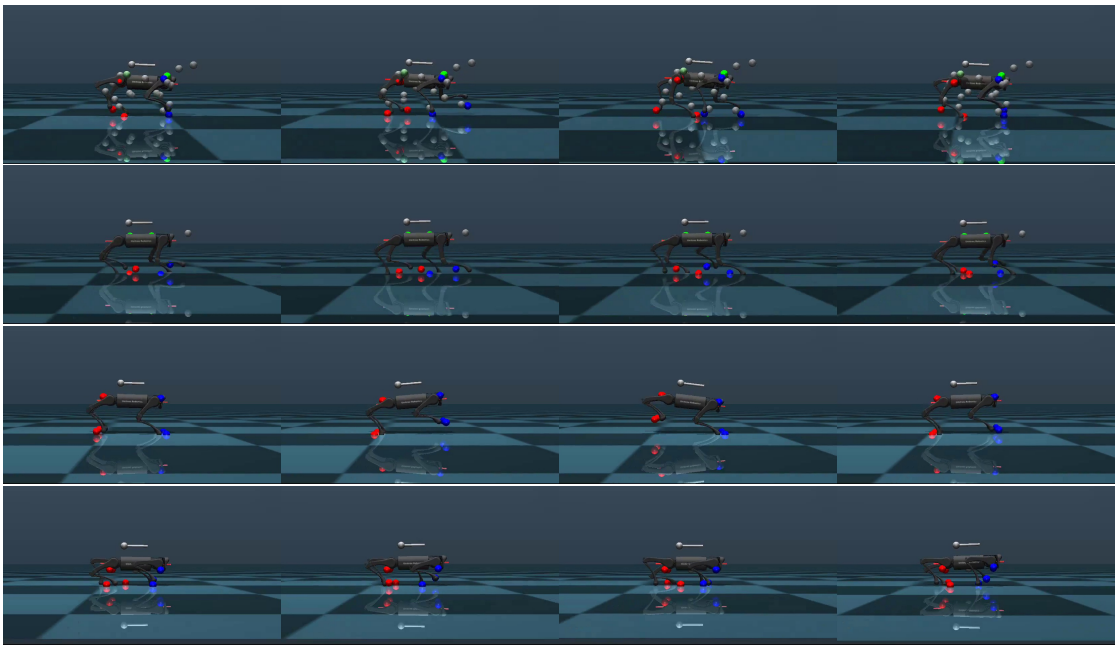


Figure A.1.: Results from Motion Retargeting on Unitree A1: From top to bottom, the retargeted motions are dog trot, horse trot, mpc bound, and solo8 crawl

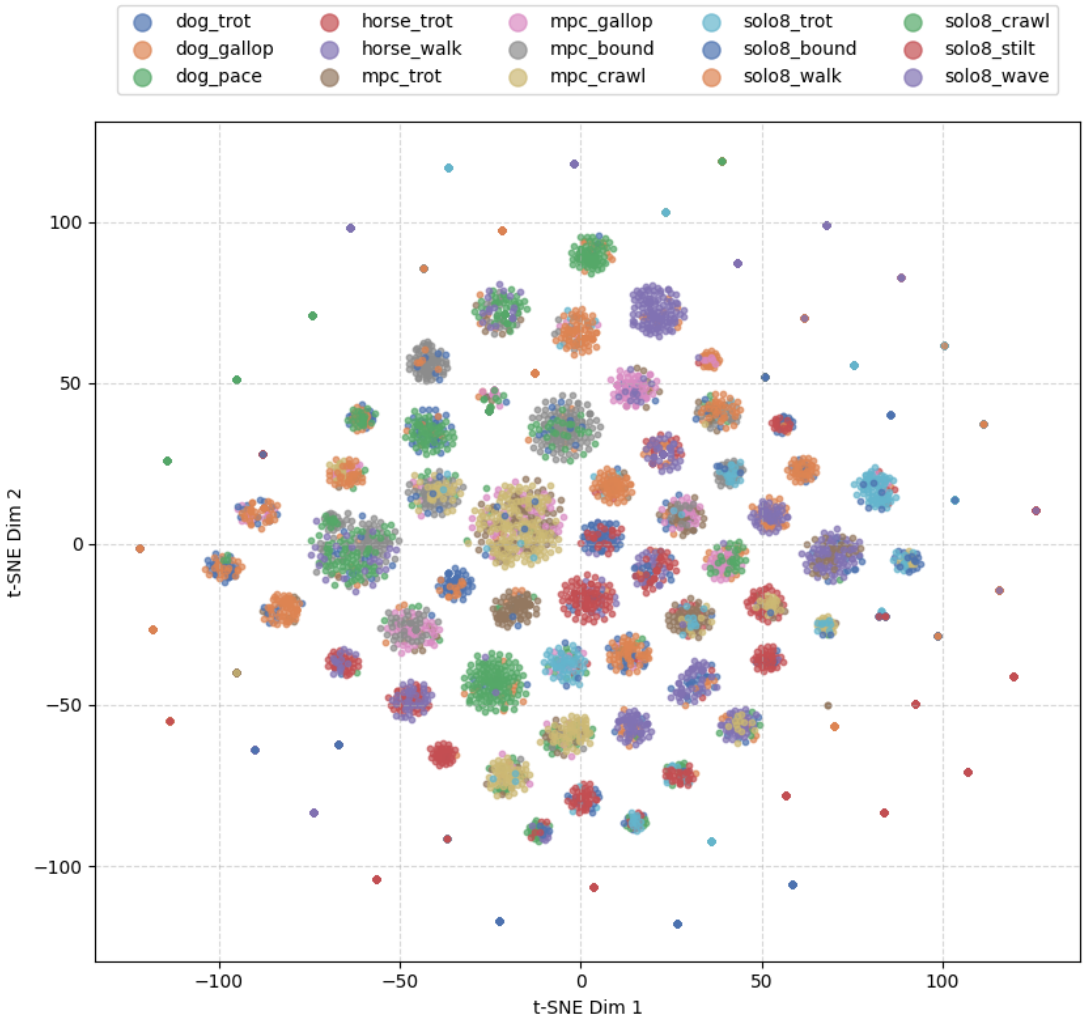


Figure A.2.: t-SNE Visualization of Latent Space with One Codebook

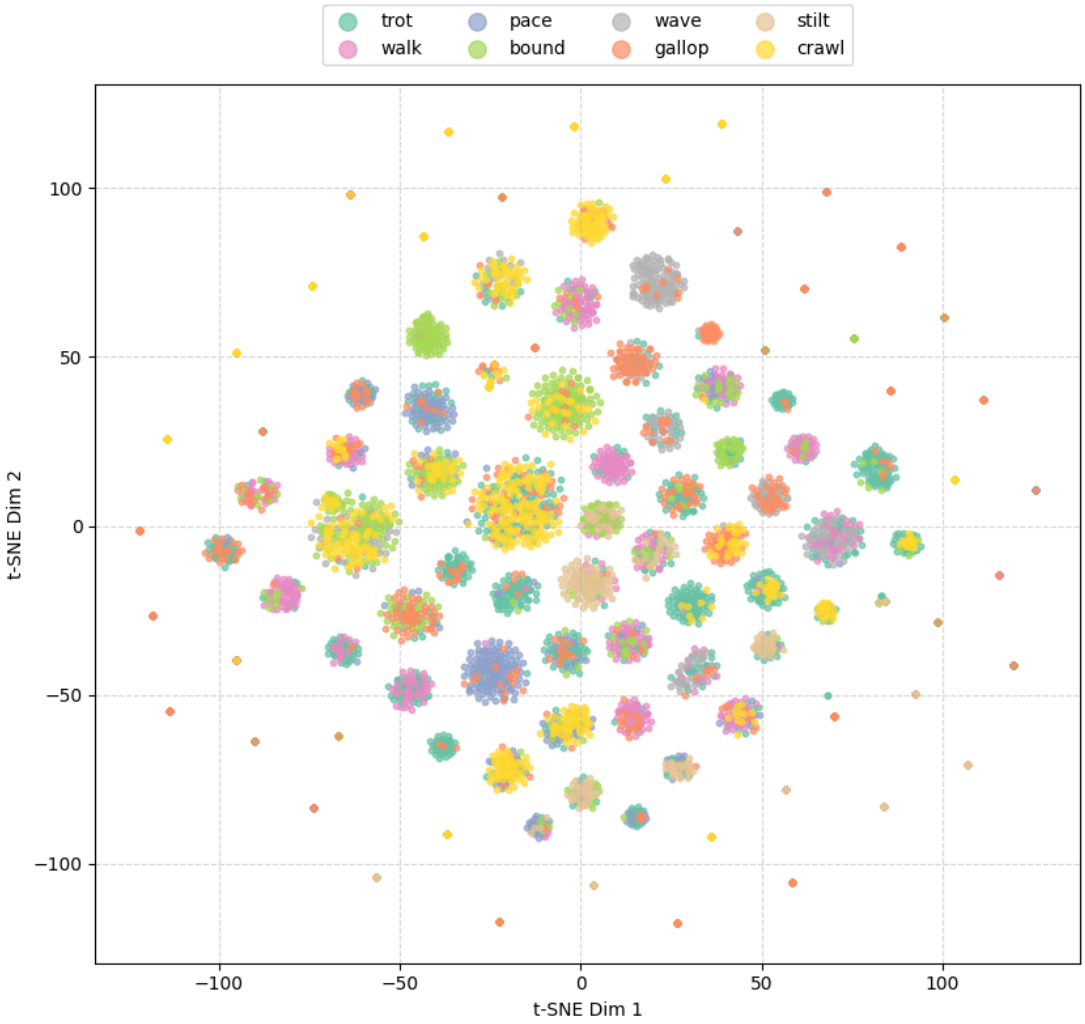


Figure A.3.: t-SNE Visualization of the Latent Space Using a Single Codebook, Organized by Gaits.

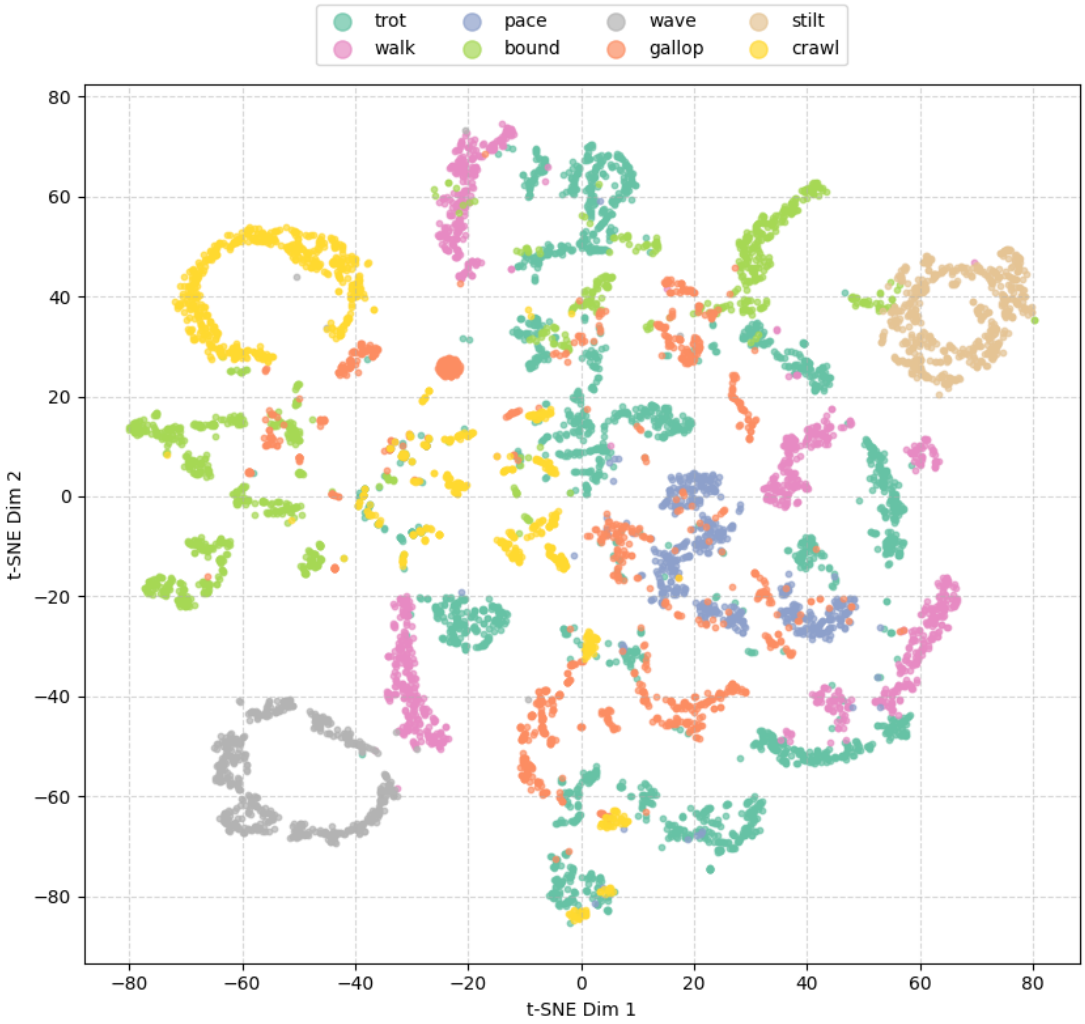


Figure A.4.: t-SNE Visualization of the Latent Space Using Four Codebooks, Organized by Gaits.

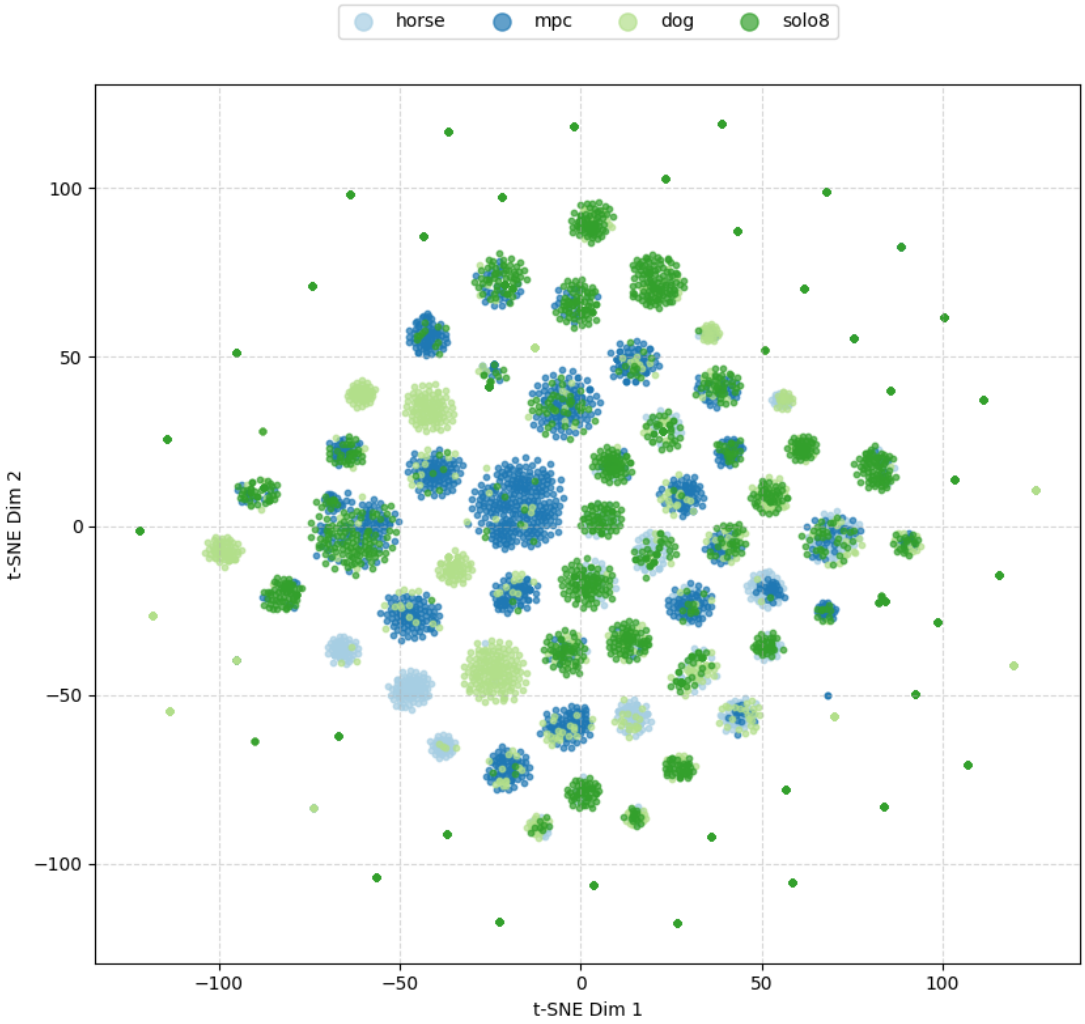


Figure A.5.: t-SNE Visualization of the Latent Space Using a Single Codebook, Organized by Sources.

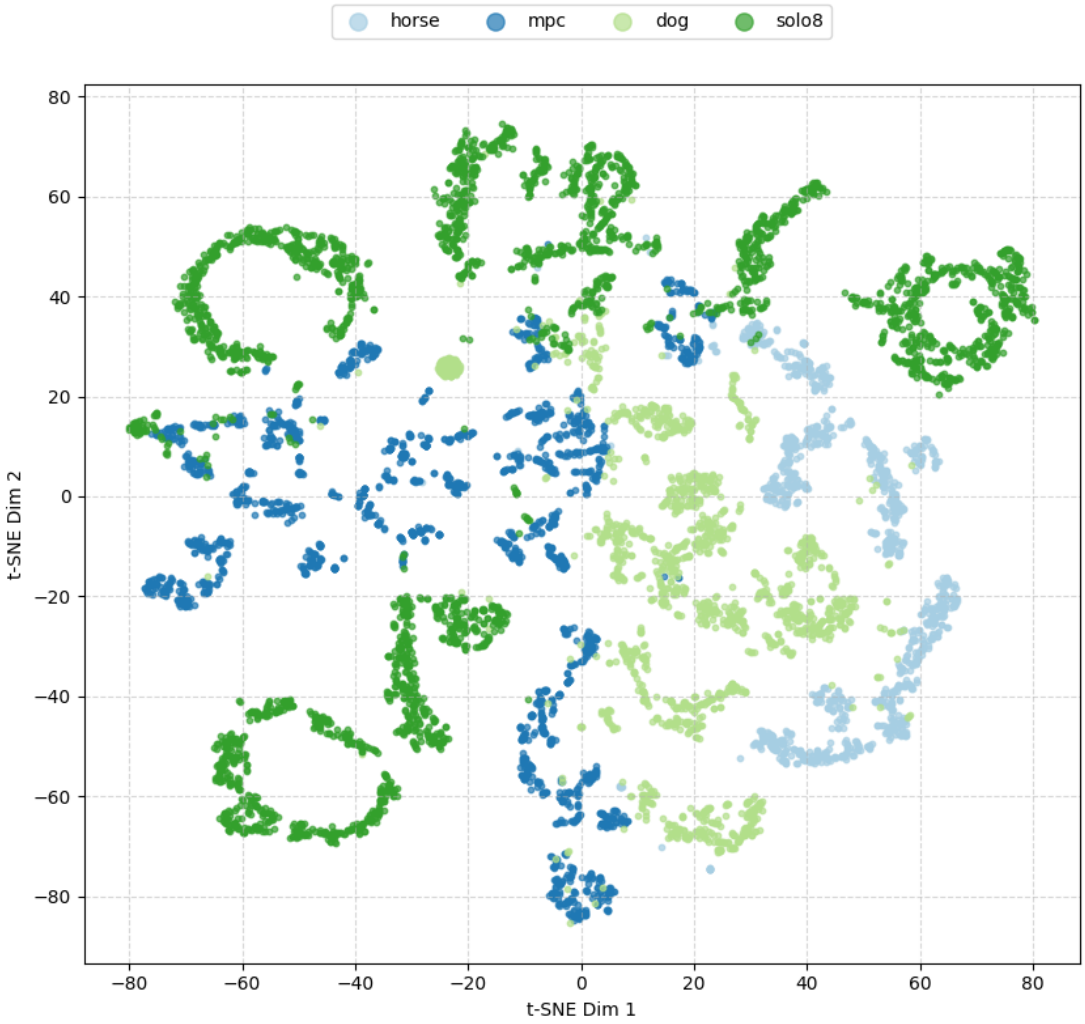


Figure A.6.: t-SNE Visualization of the Latent Space with Four Codebooks, Categorized by Data Sources: Dog, Horse, MPC, and Solo8.

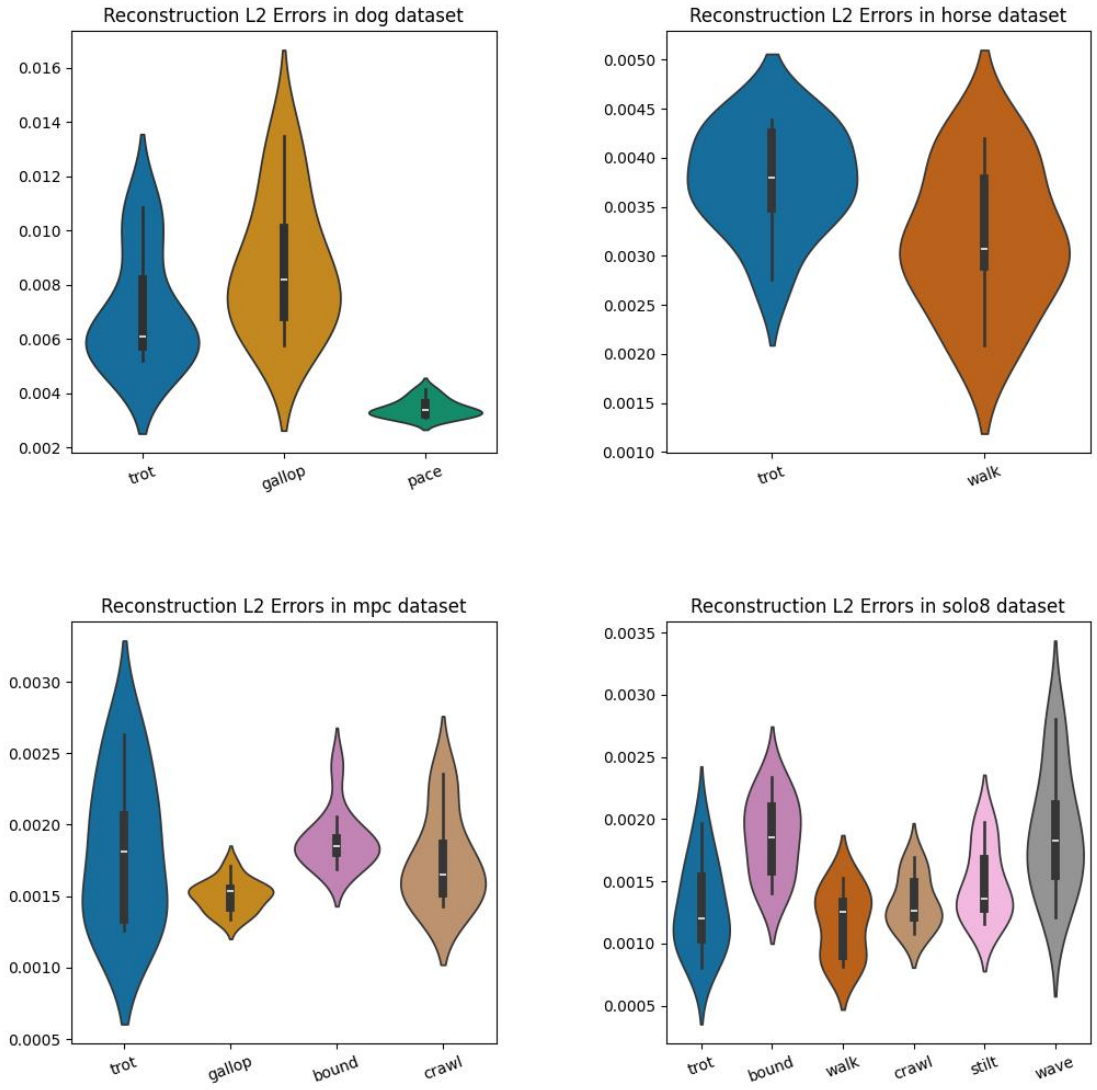


Figure A.7.: Reconstruction L2 Error from model with EMA by Gait and Motion Source.

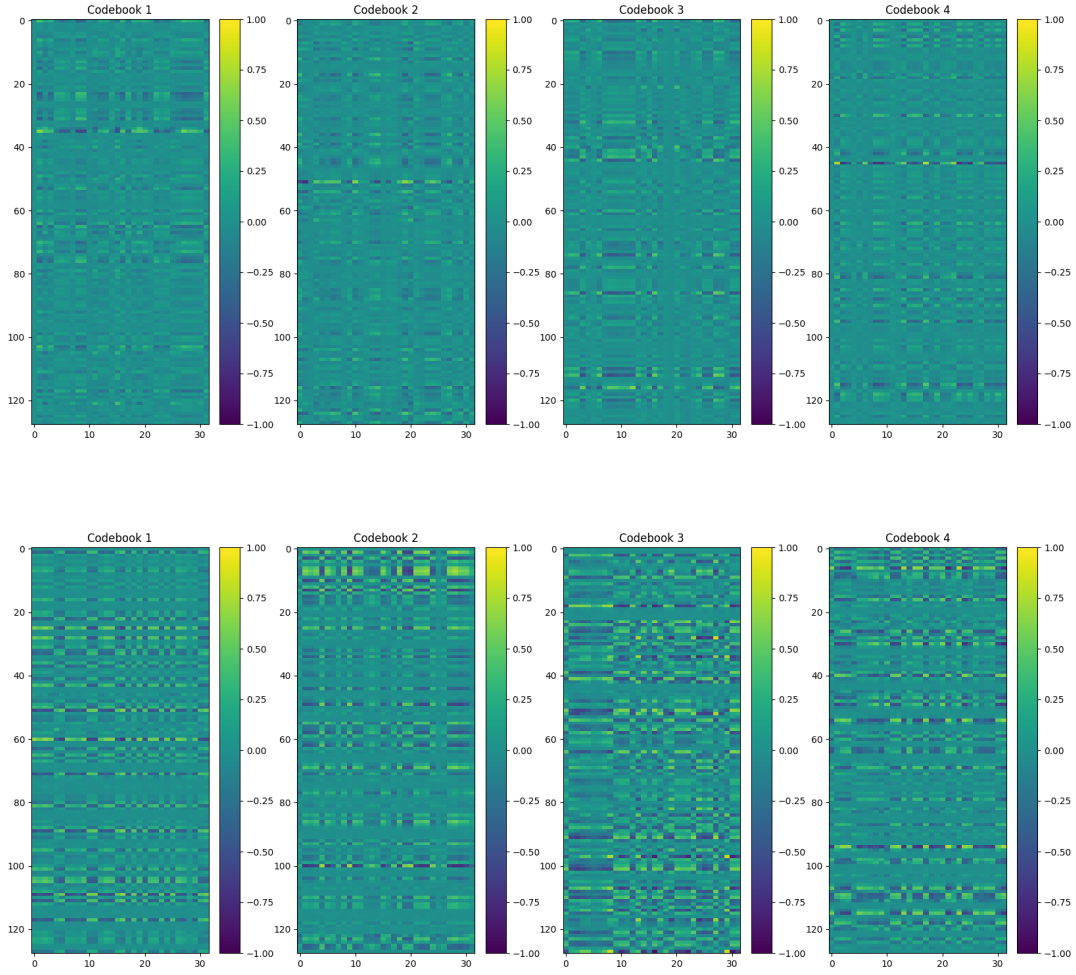


Figure A.8.: Visualization of codes in different codebooks: with EMA (above) and without EMA (below).

Hyperparameters	**VQ-VAE**	**Gen**
Number of training epochs	400	500
Learning rate for the optimizer	$1e^{-3}$	$1e^{-3}$
Batch size for training	32	32
Batch size for validation	4	4
Dropout rate for the model	0.1	0.1
Number of Blocks in the model	2	1
Embedding dimension	128	128
Number of attention heads	2	4
Block size for the model	1024	1024
Dropout rate for attention	0.3	0.3
Positional encoding for all Blocks	True	True
Number of embeddings in the codebook	512	–
Dimension of each embedding in the codebook	128	–
Number of codebooks	4	–
Autoregressive head enabled	–	True

Table A.1.: Summary of Configuration Parameters for VQ-VAE and Generator

Parameter	Value
Total timesteps	$1,000,000,000$
Number of steps per rollout	$2,720$
Minibatch size	$10,880$
Number of epochs	5
Number of Environments	16
Start learning rate	0.0004
End learning rate	0.0
Entropy coefficient	0.0
Discount factor	0.99
GAE lambda	0.9
Critic coefficient	1.0
Maximum gradient norm	5.0
Initial action standard deviation	1.0
Clip range for action standard deviation	$1e^{-8}$, 2.0
Clip range for action mean	$-10.0, 10.0$ (Before applying σ_a)
Clip range	0.1
Kp	20
Kd	0.5

Table A.2.: Summary of PPO Training Parameters

Term	Equation
Joint accelerations penalty	$-\lvert\ddot{q}\rvert^2$
Joint torques penalty	$-\lvert\tau\rvert^2$
Action rate penalty	$-\lvert\dot{a}\rvert^2$
Collisions penalty	$-n_{\text{collisions}}$

Table A.3.: Penalty used in the reward function.

Observation Space	Dimensions
Joint Positions	12
Joint Velocities	12
Current Actions	12
Trunk Linear Velocities	3
Trunk Angular Velocities	3
Projected Gravity	3
Height	1
Goal Trunk Linear Velocity	3
Goal Trunk Quaternion	4
Goal Trunk Angular Velocities	3
Goal Joint Positions	12
Goal Joint Velocities	12
Goal Relative Toe Positions	12
One-Hot Source Encoder	4
One-Hot Gait Encoder	8

Table A.4.: Observation Space.The height and trunk linear velocities are excluded from the observation space during actor training.

Term	Min	Max
Friction Tangential	0.001	2.0
Friction Torsional Ground	0.00001	0.01
Friction Torsional Feet	0.0001	0.04
Friction Rolling Ground	0.00001	0.0002
Friction Rolling Feet	0.00001	0.02
Damping	30	130
Stiffness	500	1500
Gravity	8.81	10.81
Add Trunk Mass	−2.0	2.0
Add Com Displacement	−0.01	0.01
Foot Size	0.020	0.024
Joint Damping	0.0	2.0
Joint Armature	0.0008	0.05
Joint Stiffness	0.0	2.0
Joint Friction	0.0	1.0

Table A.5.: Domain Randomization Parameters.