



POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Advancing Behavioral Compression in Unsupervised Reinforcement Learning

TESI DI LAUREA MAGISTRALE IN

COMPUTER ENGINEERING - INGEGNERIA INFORMATICA E DELL'INFORMAZIONE

Andrea Fraschini, 10725610

Advisor:

Prof. Marcello Restelli

Co-advisors:

Davide Tenedini

Academic year:

2024-2025

Abstract: The high dimensionality and redundancy of the parameter space in Deep Reinforcement Learning (DRL) present significant obstacles to efficient policy optimization and exploration. This thesis explores a novel unsupervised framework that decouples behavioral discovery from task-specific optimization by identifying and compressing the underlying behavioral manifold of an agent acting in an environment.

The methodology consists of a three-stage pipeline. First, we derive an information-theoretic uniqueness metric to curate a diverse population of unsupervised policies. Second, we employ an Autoencoder architecture to map the high-dimensional parameter space into a compact latent representation. To ensure the latent geometry reflects behavioral similarity, the model is trained using a mixture-occupancy matching objective to align global state-visitation distributions. Finally, we perform task-specific optimization within the learned manifold using Policy Gradient with Parameter-based Exploration (PGPE).

Empirical evaluations across classical control and MuJoCo robotic benchmarks demonstrate that the proposed framework achieves superior sample efficiency and optimization stability compared to standard DRL baselines. Our results show that the learned manifolds are topologically smooth and largely invariant to the size of the underlying policy networks. However, while the manifold provides a robust search space, its performance is ultimately bounded by the behavioral support of the initial dataset of policies. This work provides a foundation for more efficient task adaptation by shifting the focus from redundant parameters to meaningful behavioral representations.

Key-words: unsupervised reinforcement learning, policy compression, behavioral manifold, state-occupancy matching

1. Introduction

The long-standing quest of Artificial Intelligence (AI) is the development of autonomous agents capable of perceiving, reasoning, and acting within complex, uncertain environments to achieve diverse objectives. For decades, the primary bottleneck in this pursuit was the *representation gap*: the difficulty of translating raw, high-dimensional sensory input into meaningful features suitable for decision-making. Historically, this required

manual feature engineering, where domain experts designed heuristics to extract relevant information from the original data that could be used effectively by models.

The advent of Deep Learning (DL) fundamentally shifted this paradigm by enabling the automatic extraction of features from raw data. By leveraging multi-layered (deep) neural architectures, DL allows for the discovery of representations that compress complex input spaces into lower-dimensional manifolds, preserving essential information while discarding task-irrelevant noise [17]. This revolution has yielded unprecedented success in computer vision [15], speech recognition [13], natural language processing [19, 35], and object detection [26].

Specifically, the integration of DL with Reinforcement Learning (RL), known as Deep Reinforcement Learning (DRL), has enabled agents to tackle problems in high-dimensional continuous scenarios. From mastering adversarial games like Go at superhuman levels [30] to enabling complex autonomous navigation [24], DRL has helped tackle problems in complex settings that were previously intractable using simpler linear models and tabular approaches [20, 28].

However, despite these milestones, modern DRL remains characterized by significant brittleness and inefficiency. A DRL policy trained for a specific task often fails to generalize even to subtle environmental perturbations, such as variations in sensor noise or task parameters, usually necessitating retraining from scratch. Moreover, many state-of-the-art algorithms require large amounts of interactions with the environment, making them inefficient and impractical in real-world scenarios where data collection is expensive in terms of both time and resources.

1.1. Motivation and Goal

A widespread use of neural networks in DRL is as high-dimensional parametric models for policy representation in Policy Gradient (PG) methods. However, this high dimensionality presents notable challenges. One of the most significant issues is the high sample complexity, meaning a substantial amount of data is required to achieve an optimal set of parameters. Moreover, due to the redundant nature of neural networks, many different parameter configurations can lead to policies that exhibit very similar behaviors. This ‘over-parametrization’ makes it difficult to use neural networks for modeling policies that can adapt effectively to various environments, as the mapping between parameter updates and behavioral changes is non-linear and difficult to understand. More fundamentally, a limiting characteristic of DRL methods is that the discovery of behaviors is tied to a specific extrinsic reward signal. Consequently, if an agent focuses only on what is necessary for the task, it may overlook the vast range of possibilities that exist within the environment.

Building on recent advances in policy space compression [33], the premise of this thesis is that the diversity of behaviors an agent can exhibit is fundamentally constrained by the environment’s dynamics, regardless of the policy network’s size. We hypothesize that the manifold of realizable behaviors is intrinsically low-dimensional and that its structure remains largely invariant to increased network capacity. Based on this intuition, we propose a novel Unsupervised Learning training framework that identifies this behavioral manifold to shift optimization from a redundant, high-dimensional parameter space to a compact and informative latent representation in a completely unsupervised manner, meaning it does not rely on reward signals. This allows the agent to leverage past interactions to solve a variety of tasks more efficiently, effectively decoupling behavioral discovery from task-specific optimization.

1.2. Main Contributions

The main contributions of this thesis are the following:

- **Design and empirical validation of an Unsupervised RL pipeline.** By decoupling task-agnostic behavioral discovery from task-specific optimization, this framework enables the identification of a low-dimensional behavioral manifold within a high-dimensional parameter space, reducing the sample complexity of downstream Reinforcement Learning tasks.
- **Derivation of an information-theoretic metric for behavioral uniqueness.** Grounded in the mathematical decomposition of the entropy of a mixture occupancy distribution, this metric quantifies the marginal contribution of individual policies to the global state-space coverage of a population of policies.
- **Development of a mixture-matching objective for Autoencoders.** Unlike standard reconstruction losses, this approach utilizes the Kullback-Leibler Divergence to compare the collective state occupancy distributions of entire populations of policies. By leveraging a non-parametric importance sampling estimator, this loss function enables the alignment of global behavioral manifolds while maintaining computational efficiency by recycling trajectory data.

1.3. Related Works

Research for efficient behavior representation and discovery spans several sub-fields of Reinforcement Learning. Our work sits at the intersection of policy compression, unsupervised behavior discovery, and occupancy matching.

Policy Compression Policy compression, more generally known as neural network compression, traditionally focuses on reducing the complexity of neural architectures without degrading performance. *Policy Distillation* [27], for instance, is used to transfer knowledge from a teacher to a smaller student network. Recent research has introduced frameworks for reward-free policy space compression [22], demonstrating that a limited set of policies can effectively cover the manifold of achievable behaviors in an environment. Our work builds on this premise, leveraging recent advances in parameter space compression [33] to map high-dimensional configurations into a compact, continuous latent manifold rather than a discrete set of representative policies.

Task-Agnostic Exploration and Unsupervised RL A key challenge in DRL is the dependency on extrinsic rewards, which often limits an agent’s ability to generalize. To address this, some unsupervised methods seek to learn useful behaviors before a specific goal is defined in order to find a robust task-agnostic starting point for downstream optimization. Among others, *Maximum Entropy POLicy* (MEPOL, [23]) utilizes a max-entropy objective to drive an agent toward maximum state-space coverage. Similarly, information-theoretic frameworks like *Diversity Is All You Need* (DIAYN, [5]) aim to discover diverse skills by maximizing the mutual information between states and latent variables. While these methods focus on finding good starting points, our framework seeks to identify a global behavioral manifold that facilitates efficient adaptation across a variety of tasks.

Occupancy Matching and Imitation Learning Comparing behaviors mathematically often involves analyzing their state occupancy distributions. This is the foundation of Imitation Learning (IL) techniques, which seek to minimize the divergence between an agent’s behavior and a target distribution. Many of these methods employ *Generative Adversarial Networks* (GANs, [8]), such as the *Generative Adversarial Imitation Learning* (GAIL, [14]) framework. In particular, *Wasserstein-GANs* (WGANs, [2]) have been successfully applied to behavior matching tasks [9, 37]. Our contribution extends these concepts by developing a mixture-matching objective for autoencoders. Unlike standard IL that matches a single expert, our approach aligns the collective state occupancy of entire populations, utilizing the KL-divergence between mixtures to measure differences in behavioral spaces.

2. Preliminaries

Notation Calligraphic letters (e.g. $\mathcal{S}$) are used to denote sets, $|\mathcal{S}|$ is used to indicate the cardinality of set $\mathcal{S}$, and $\Delta(\mathcal{S})$ is used to indicate the probability simplex over $\mathcal{S}$, defined as the set of all probability distributions on $\mathcal{S}$. If $\mathcal{S}$ is finite, this corresponds to: $\Delta(\mathcal{S}) = \{p : \mathcal{S} \rightarrow [0, 1] \mid \sum_{s \in \mathcal{S}} p(s) = 1\}$. In the case of continuous spaces, $\Delta(\mathcal{S})$ denotes the set of probability density functions (or more generally, probability measures) over $\mathcal{S}$. $\Pr(E)$ is used to indicate the generic probability of event E .

2.1. Controlled Markov Processes

The interaction model with the environment is a straightforward finite-horizon *Controlled Markov Process* (CMP), which can be understood as a Markov Decision Process (MDP, [25]) without a reward function. This choice is informed by the fact that the methodologies utilized, except for the final optimization algorithm, are all unsupervised and thus independent of any reward signal.

Formally, a CMP is defined by the tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, \nu, T)$, where:

- $\mathcal{S}$ is the set of all possible states (state space).
- $\mathcal{A}$ is the set of all possible actions (action space).
- $P : \mathcal{S} \times \mathcal{A} \rightarrow \Delta(\mathcal{S})$ is the transition probability distribution.
- $\nu \in \Delta(\mathcal{S})$ is the initial state distribution.
- $T \in \mathbb{N}$ is the time horizon length.

The interactions occur in episodes. At the start of each episode, an initial state s_0 is drawn from the distribution ν . At each time step t , the agent observes the current state s_t , selects an action a_t according to its policy π ,

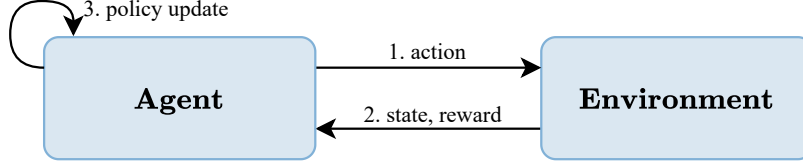


Figure 1: Agent-Environment interaction. The agent, given the current state, takes an action and observes the next state and a reward value; it then uses this information to update its behavior (policy).

and the environment transitions to a new state s_{t+1} sampled according to $s_{t+1} \sim P(\cdot | s_t, a_t)$. This process is repeated until the terminal state s_T is reached, as shown by steps 1 and 2 in Figure 1.

The agent’s behavior is governed by a (stochastic) policy $\pi : \mathcal{S} \rightarrow \Delta(\mathcal{A})$. We denote the conditional probability (or probability density in continuous environments) of choosing action a given state s as $\pi(a | s)$.

When an agent equipped with a policy π acts within a CMP, it generates a *trajectory* of states and actions for each episode, denoted by the sequence $\tau = (s_0, a_0, s_1, a_1, \dots, s_{T-1}, a_{T-1}, s_T)$. The agent, by collecting trajectories over episodes with policy π , also induces a marginal probability distribution over states called the *state occupancy distribution*, defined as the time-averaged probability of observing state s across a trajectory:

$$d_\pi(s) = \frac{1}{T} \sum_{t=0}^T \Pr(s_t = s | \pi).$$

In scenarios involving a population of M policies $\Psi = \{\pi_1, \dots, \pi_M\}$ interacting independently with the environment, we consider the *mixture occupancy distribution* d_Ψ . If a policy is sampled uniformly from Ψ to collect a trajectory, then the mixture occupancy distribution is:

$$d_\Psi(s) = \frac{1}{M} \sum_{i=1}^M d_{\pi_i}(s).$$

This mixture distribution characterizes the overall state-space coverage achieved by the population Ψ , reflecting the probability of encountering state s when any policy from the population is deployed with equal probability.

2.2. Policy Optimization and PGPE

In the context of Markov Decision Processes, the goal is to identify a policy π that maximizes a performance objective $J(\pi)$. This objective is typically the expected cumulative reward, defined over timesteps, over the trajectories induced by the interaction between the agent and the environment. Formally, we define the objective function as:

$$J(\pi) = \mathbb{E}_{\tau \sim \rho(\cdot | \pi)} \left[\sum_{t=0}^{T-1} R(s_t, a_t) \right], \quad \rho(\tau | \pi) = \nu(s_0) \prod_{t=0}^{T-1} P(s_{t+1} | s_t, a_t) \pi(a_t | s_t),$$

where $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the reward function and $\rho(\tau | \pi)$ is the probability measure over trajectories induced by policy π . The update to the agent’s behavior (policy) to maximize $J(\pi)$ corresponds to step 3 in Figure 1.

While several approaches exist, we consider strategies that focus on optimizing the policy itself, known as *Policy Optimization* (PO, [4]) methods. In this framework, policies are represented by parameter vectors $\theta \in \Theta \subseteq \mathbb{R}^n$ (e.g., weights of a neural network in DRL), denoted by $\pi_\theta \in \Pi_\theta$, where Θ is used to indicate the *parameter space*, and Π_θ the *policy space*. Consequently, the problem becomes a maximization of the objective function with respect to the parameters:

$$\theta^* = \arg \max_{\theta \in \Theta} J(\theta) = \arg \max_{\theta \in \Theta} \mathbb{E}_{\tau \sim \rho(\cdot | \pi_\theta)} \left[\sum_{t=0}^{T-1} R(s_t, a_t) \right].$$

Additionally, if $J(\theta)$ is differentiable, we can use gradient-ascent methods to find a local maximum by applying the update rule $\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\theta)$, where α is the learning rate. A fundamental result in this domain is the *Policy Gradient Theorem* [31], which provides an expression for the gradient that does not require explicit knowledge of the environmental dynamics P :

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \rho(\cdot | \pi_\theta)} \left[\left(\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t | s_t) \right) \left(\sum_{t=0}^{T-1} R(s_t, a_t) \right) \right].$$

Algorithms that optimize the objective function by performing gradient updates directly on the policy parameters are known as *Policy Gradient* (PG, [31, 36]) methods.

While standard PG methods typically rely on stochastic policies to explore the action space $\mathcal{A}$, an alternative approach is to perform exploration directly in the parameter space Θ . A notable method in this category is *Policy Gradient with Parameter-based Exploration* (PGPE, [29]). PGPE introduces a *hyper-policy* μ_ϕ , which is a probability measure over the parameter space Θ , parameterized by ϕ . Just as a stochastic policy π_θ defines a distribution over actions for a given state, the hyper-policy $\mu_\phi(\theta)$ serves as a generative model for the policy weights θ . This allows the agent to explore the environment by sampling entire sets of parameters, and thus possibly distinct behaviors, for each episode, which can be more effective than per-step action noise.

In practice, the hyper-policy is often modeled as a multivariate Gaussian distribution: $\mu_\phi(\theta) = \mathcal{N}(\theta \mid \mathbf{m}, \Sigma)$, where the hyper-parameters $\phi = \{\mathbf{m}, \Sigma\}$ represent the mean and the covariance of the distribution.

Under this framework, the objective function is redefined as the expected cumulative reward over both the parameter distribution and the resulting trajectories. The gradient of the objective with respect to ϕ is derived as:

$$\nabla_\phi J(\phi) = \mathbb{E}_{\theta \sim \mu_\phi(\cdot)} [\mathbb{E}_{\tau \sim \rho(\cdot \mid \pi_\theta)} [R(\tau)] \nabla_\phi \log \mu_\phi(\theta)],$$

where $R(\tau) = \sum_{t=0}^{T-1} R(s_t, a_t)$ denotes the cumulative reward of a trajectory τ . This formulation also has the advantage of allowing for the optimization of the distribution of policies without requiring the differentiability of the underlying policy π_θ with respect to its weights, as the gradient is taken with respect to the hyperparameters ϕ .

2.3. Unsupervised Reinforcement Learning

While the aforementioned Policy Optimization framework relies on the existence of an *extrinsic* reward function R to guide the search in the parameter space, the *Unsupervised Reinforcement Learning* (URL, [16]) framework shifts the focus from task-specific optimization to task-agnostic interaction within a CMP as described in Section 2.1. In this paradigm, the agent’s objective is decoupled from a specific reward function and is instead driven by *intrinsic* motivation, an internal reward strategy designed to encourage the discovery of the environment’s underlying structure or the acquisition of a diverse set of behaviors. The URL process is organized into two distinct phases:

1. **Unsupervised Pre-training:** The agent interacts with the CMP to maximize an intrinsic objective (e.g., state-space coverage) to produce a rich representation of the environment or a set of diverse policies without any knowledge of the downstream tasks.
2. **Supervised Fine-tuning:** Given an arbitrary task from a set of unknown tasks, the agent leverages the knowledge and/or representations acquired during pre-training to accelerate the optimization process.

The efficacy of a URL algorithm is measured by its *adaptation efficiency*: the speed and performance with which it can solve an unknown set of tasks during the fine-tuning phase. This thesis operates within this framework, specifically focusing on how the unsupervised compression of the policy space Π_θ can yield a low-dimensional latent space $\mathcal{Z}$ that is more conducive to efficient policy optimization than the original, high-dimensional parameter space Θ .

2.4. Manifold Hypothesis and Autoencoder Models

A widely accepted thesis in Deep Learning is the so-called *Manifold Hypothesis* [3, 6]. This concept states that most real-world high-dimensional data actually exists within a low-dimensional subspace referred to as a *latent manifold* or *latent space*. This means that instead of using the many coordinates of the original space, it’s possible to describe the data using fewer coordinates of the underlying latent manifold.

This hypothesis has been empirically validated in many situations. In image processing, for example, while a 1024×1024 grayscale image resides in a 2^{20} -dimensional space, the set of natural images (e.g., landscapes, human faces) occupies a negligible fraction of this space. The manifold of human faces, for instance, could be parametrized by a few variables representing pose, background, skin tone, and expression rather than individual pixel intensities.

The most prominent architecture that exploits the manifold hypothesis is the *Autoencoder* (AE, [12]). An autoencoder is a neural network designed to learn an identity mapping in an unsupervised manner.

Let $\mathcal{X} \subseteq \mathbb{R}^n$ denote the high-dimensional input space and $\mathcal{Z} \subseteq \mathbb{R}^k$ denote the latent space, with $k \ll n$. This dimensionality constraint creates a *bottleneck* that forces the model to discard noise and capture only the most salient features of the input data.

The architecture is composed of two parts:

1. **Encoder:** A parametric function $E_\xi : \mathcal{X} \rightarrow \mathcal{Z}$ that maps the high-dimensional input $\mathbf{x}$ to a lower-dimensional latent code $\mathbf{z}$; effectively performing dimensionality reduction. It is defined by a set of

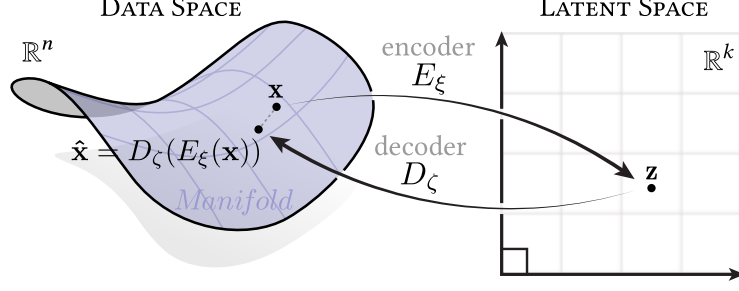


Figure 2: Representation of Autoencoder Spaces and Data Manifold.

learnable parameters ξ such that $z = E_\xi(x)$.

2. **Decoder:** A parametric function $D_\zeta : \mathcal{Z} \rightarrow \mathcal{X}$ that attempts to reconstruct the original input from the latent code. It is defined by a set of learnable parameters ζ such that $D_\zeta(E_\xi(x)) \approx x$.

A graphical representation of how autoencoders are able to identify and map the data manifold is shown in Figure 2.

The objective of AE training is to find the optimal set of parameters (ξ^*, ζ^*) that minimizes the reconstruction error over a dataset $\mathcal{D} = \{x\}_{i=1}^N$. Formally, the optimization problem is:

$$(\xi^*, \zeta^*) = \arg \min_{\xi, \zeta} \mathbb{E}_{x \sim \mathcal{D}} [\mathcal{L}(x, D_\zeta(E_\xi(x)))], \quad (1)$$

where $\mathcal{L}$ is a loss function. Usually, $\mathcal{L}$ is chosen as a distance metric in the input space $\mathcal{X}$, such as the Euclidean distance or the Mean Squared Error.

In the context of this thesis, the input x corresponds to the behavior of a policy, specifically its state occupancy distribution d_π . Since these occupancies are probability distributions, the standard Euclidean distance is insufficient. Instead, we utilize a divergence-based loss function D , such as the Kullback-Leibler (KL) divergence, to capture the manifold of behaviors that an agent can exhibit within the environment.

2.5. Non-Parametric Divergence Estimator

As defined in Section 2.1, the state occupancy distribution d_π represents the marginal probability of visiting a state under policy π . In practice, an analytical form for d_π is generally intractable. This is because computing it requires evaluating the integral over all possible trajectories, which involves the environment’s transition probability P . In most RL scenarios, these dynamics are unknown or too complex to model analytically.

Consequently, divergence measures between occupancies, such as the Kullback-Leibler divergence $D_{\text{KL}}(d_\pi \parallel d_{\pi'})$, must be estimated using *particles* (states) obtained through interactions with the environment.

However, standard particle-based estimators typically require fresh samples from both the source distribution d_π and the target distribution $d_{\pi'}$. In an optimization context, where one distribution is usually parametrized and subject to frequent updates, the requirement to resample from the environment for every evaluation of the divergence leads to a prohibitive sample complexity. In a URL or Policy Compression framework that deals with populations of policies Ψ and associated mixture occupancy distributions d_Ψ , this problem is further exacerbated.

To mitigate this, we consider a non-parametric k -nearest neighbors (k -NN) importance sampling estimator [1]. This approach allows us to estimate the divergence between a sampling distribution f and a target distribution f' as follows:

$$\hat{D}_{\text{KL}}(f \parallel f') = \frac{1}{N} \sum_{i=1}^N \ln \frac{k/N}{\sum_{j \in \mathcal{N}_i^k} w_j}, \quad (2)$$

where N is the number of particles, $\mathcal{N}_i^k$ is the set of indices of the k -nearest neighbors of sample x_i , and w_j are the self normalized importance weights:

$$w_j = \frac{f'(x_j)/f(x_j)}{\sum_{n=1}^N f'(x_n)/f(x_n)}. \quad (3)$$

The main advantage of this estimator in our framework is its sample efficiency. Thanks to the importance weights w_j , we can use a fixed set of particles collected from a known sampling distribution f (e.g., a mixture

occupancy d_Ψ) to evaluate the divergence for a target distribution f' , effectively separating particle collection from divergence estimation. This property is particularly beneficial in iterative optimization settings, as it enables the reuse of collected trajectories across multiple evaluations. As a result, it's possible to perform updates to a parametric model without expensive interactions with the environment.

3. Problem Formulation

As established in Section 2.2, standard Policy Optimization (PO) seeks an optimal parameter vector $\theta^* \in \Theta$ that maximizes a cumulative reward. However, the objective function fundamentally depends on the state-action occupancy distribution. To effectively solve a task, an agent must identify and visit the states and actions that yield the highest rewards. This suggests that the primary object of interest is not the parameter vector θ^* itself, but rather the *behavior* it induces, formally represented by the state occupancy distribution $d_{\pi_{\theta^*}}$.

Shifting the focus from the parameter space Θ to the space of behaviors addresses several issues in Deep Reinforcement Learning. First, the mapping from parameters to behaviors is highly redundant: many distinct θ configurations may induce nearly identical distributions over actions. Furthermore, even different action distributions can lead to comparable state-occupancy distributions due to the constraints imposed by the environment transition dynamics P .

Following the two-stage framework of *Latent Policy Compression and Optimization* introduced by [33], we address these challenges by extending the Manifold Hypothesis to the domain of Deep Reinforcement Learning. Under this premise, the manifold of realizable behaviors is assumed to be intrinsically low-dimensional and largely invariant to the policy network's capacity. Increasing the number of parameters primarily introduces redundancy into the mapping rather than expanding the dimensionality of the reachable behavior space. By identifying this underlying manifold, we can shift the optimization problem from the redundant parameter space $\Theta \subseteq \mathbb{R}^n$ to a compact, more informative *latent space* $\mathcal{Z} \subseteq \mathbb{R}^k$, where $k \ll n$.

To achieve this shift from parameter space to latent space, [33] introduce a generative function $g : \mathcal{Z} \rightarrow \Theta$ that maps a low-dimensional latent code $z \in \mathcal{Z}$ to a policy parameter vector $\theta = g(z)$, allowing any policy within the discovered manifold to be expressed as $\pi_z = \pi_{g(z)}$. While the original formulation minimizes an action-wise divergence between policies, we instead minimize the divergence between mixture occupancy distributions. Consequently, we redefine the two stages of the process as follows.

Latent Behavior Compression In this stage, the objective is to identify an optimal generative function g^* that minimizes the divergence D between the behavior of an original population of parametric policies $\Psi \subset \Pi_\theta$ and its latent reconstruction. Formally, we seek g^* such that:

$$\exists Z \in \mathcal{Z}^{|\Psi|} : g^* = \arg \min_g D(d_\Psi \parallel d_{g(Z)}), \quad (4)$$

where $Z = \{z_1, \dots, z_{|\Psi|}\}$ is the set of latent codes corresponding to the policies in Ψ , and $g(Z) = \{g(z) \mid z \in Z\}$ denotes the set of reconstructed policy parameters. The terms d_Ψ and $d_{g(Z)}$ correspond to the mixture occupancy distributions of the original and reconstructed populations, respectively.

Latent Behavior Optimization In this second stage, once the generative function g^* is identified and a task-specific reward function R is given, the agent performs a supervised search within the learned manifold to find an optimal behavior that maximizes the objective function $J^R(\theta = g(z))$. The problem of finding an optimal parameter vector θ^* is thus reformulated as identifying the optimal latent code z^* :

$$z^* = \arg \max_{z \in \mathcal{Z}} J^R(g(z)). \quad (5)$$

This two-part formulation separates the discovery of the agent's behavioral repertoire (determining what the agent *can* do) from the selection of a task-optimal strategy (determining what the agent *should* do). By shifting the search to a compact, non-redundant space, this approach facilitates more efficient adaptation to new tasks and addresses the challenges associated with high-dimensional policy representations.

4. Methodology

In this section, we delineate the implementation of the updated framework for Latent Behavior Compression and Optimization. The methodology is structured as a modular pipeline designed to recover and exploit the behavioral manifold of an agent within a given Controlled Markov Process. To realize the objectives defined in the problem formulation (Section 3), the process is partitioned into three principal stages:

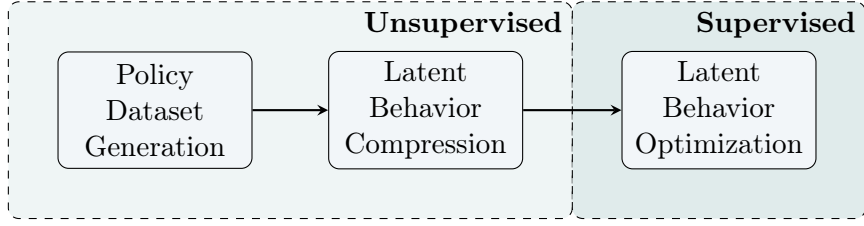


Figure 3: Pipeline of Unsupervised Compression of the Policy Space from [33].

1. **Policy Dataset Generation:** This first phase focuses on the generation of a diverse dataset of policies Ψ and their corresponding trajectories. The goal is to obtain a representative sample of the agent’s reachable behavior space through unsupervised interactions with the environment.
2. **Latent Behavior Compression:** In this stage, the high-dimensional parameter space Θ is mapped to a low-dimensional latent space $\mathcal{Z}$. We realize this through an autoencoder architecture that utilizes an occupancy-matching objective.
3. **Latent Behavior Optimization:** The final phase involves task-specific optimization. By leveraging the learned generative mapping $g : \mathcal{Z} \rightarrow \Theta$, we employ an optimization algorithm to identify the latent code z^* that maximizes the expected return for a specific downstream task.

A graphical representation of the process is shown in Figure 3.

While the theoretical framework admits various implementations for each stage, the following sections describe the specific algorithmic choices and architectural configurations utilized in this work.

4.1. Policy Dataset Generation

In order to train the autoencoder architecture, two distinct datasets are required. The primary dataset is a collection of policy parameters $\mathcal{D}_\Theta = \{\theta_1, \dots, \theta_M\}$, which represents a population of neural policies. Since we assume a fixed neural network architecture throughout this work, each parameter vector $\theta \in \Theta \subseteq \mathbb{R}^n$ entirely defines a policy π_θ . Consequently, we refer to $\mathcal{D}_\Theta$ interchangeably as a population of policies Ψ or as a dataset of weights.

This population must exhibit a broad state-space coverage to ensure that the policies are not restricted to traversing a narrow region of the state space. If the population lacked diversity, the autoencoder would fail to capture the full extent of the behavioral manifold, potentially leading to the exclusion of regions that correspond to optimal task-specific strategies. Furthermore, for training stability, the behaviors should be represented in a balanced manner within $\mathcal{D}_\Theta$. While such a population can be constructed through manual reward engineering or unsupervised exploration methods [23], we employ a straightforward yet effective filtering approach. We initially generate a large set of candidate policy parameters through random initialization and, from this initial pool, we identify and select the most representative policies by ranking them using a quantitative *uniqueness metric*.

The second dataset is a collection of trajectories $\mathcal{D}_\tau$, obtained by deploying the policies in $\mathcal{D}_\Theta$ within the environment. This dataset serves a dual purpose: first, it provides the empirical data needed to calculate the uniqueness metric for policy filtering, and second, it serves as the particle dataset required to compute the loss function used to train the autoencoder.

Scores computation: The uniqueness metric, referred to as the *score* of a policy, is designed to capture two behavioral characteristics: individual exploration capacity and relative distinctiveness with respect to the rest of the population. As established in Section 2.1, any population of policies Ψ can be treated as a mixture of behaviors, characterized by its mixture occupancy distribution d_Ψ . This distribution serves as a global descriptor of the population’s collective behavior and provides the reference frame for calculating individual scores. Formally, we define the *score* ω_i of a policy π_{θ_i} within a population Ψ as:

$$\omega_i = H(d_{\theta_i}) + D_{\text{KL}}(d_{\theta_i} \parallel d_\Psi), \quad (6)$$

where $H(d_{\theta_i})$ denotes the differential entropy of the individual state occupancy distribution d_{θ_i} associated with policy π_{θ_i} .

Equation (6) is derived from the formula for the differential entropy of the mixture occupancy distribution:

$$H(d_\Psi) = \frac{1}{M} \sum_{i=1}^M [H(d_{\theta_i}) + D_{\text{KL}}(d_{\theta_i} \parallel d_\Psi)]. \quad (7)$$

A comprehensive derivation of (7) is provided in Appendix A. It is evident from Equation (7) that the individual scores ω_i represent the marginal contribution of each policy to the total entropy (and thus the total state-space coverage) of the mixture. By selecting policies associated with higher scores, we effectively isolate behaviors that maximize both individual state-space exploration and relative diversity. This selection strategy ensures that the refined dataset preserves the maximum possible state-space coverage of the original mixture while significantly reducing its cardinality.

Since the analytical form of the occupancy d_{θ} is unknown, we first fit Gaussian Mixture Models (GMMs) to the trajectories $\mathcal{D}_{\tau}$, yielding the density estimators $\hat{d}_{\theta_i}(s)$ for every policy and $\hat{d}_{\Psi}(s)$ for the global mixture. We selected GMMs because they are especially effective at providing smooth and stable approximations of multimodal probability distributions, such as state occupancy distributions. Following this, we apply *Monte-Carlo* (MC) integration to approximate the values of H and D_{KL} .

Let $\mathcal{D}_{\theta_i} = \{s_j\}_{j=1}^N$ be a set of N particles sampled from the GMM density estimate $\hat{d}_{\theta_i}$ of the occupancy distribution of policy π_{θ_i} . The MC estimates for the differential entropy and the Kullback-Leibler divergence are then computed as:

$$\begin{aligned}\hat{H}(d_{\theta_i}) &= -\frac{1}{N} \sum_{j=1}^N \log \hat{d}_{\theta_i}(s_j), \\ \hat{D}_{\text{KL}}(d_{\theta_i} \parallel d_{\Psi}) &= \frac{1}{N} \sum_{j=1}^N \left[\log \hat{d}_{\theta_i}(s_j) - \log \hat{d}_{\Psi}(s_j) \right].\end{aligned}$$

Considerations for Robust Density Estimation To ensure that the scores ω_i are robustly estimated and maintain their discriminative power, there are two important factors that must be addressed.

First, trajectory particles are inherently temporally autocorrelated, violating the *independent and identically distributed* assumption required by GMMs for robust density estimation. To mitigate this, we employ temporal downsampling, extracting states from the trajectories at fixed intervals. This process reduces redundant information and ensures that the GMMs learn the global support of the occupancy distributions rather than overfitting the high-density paths formed by specific trajectories.

Second, the estimation of the mixture occupancy distribution d_{Ψ} must be handled carefully to avoid a *saturation effect* that can degrade the discriminative power of the scores. In a uniform mixture distribution across M policies, the KL divergence between any individual component and the mixture distribution is upper-bounded according to this inequality:

$$D_{\text{KL}}(d_{\theta_i} \parallel d_{\Psi}) \leq \log M. \quad (8)$$

A formal derivation for (8) is shown in Appendix B.

This bound shows that if $\hat{d}_{\Psi}$ is estimated using an exhaustive aggregation of trajectories from every policy in the population Ψ , the global reference distribution becomes increasingly representative of the specific policies being evaluated, and, as the population size M grows and the state space becomes more densely covered, the KL term vanishes, failing to capture the relative uniqueness of the behavior. To preserve the sensitivity of the uniqueness metric, we estimate $\hat{d}_{\Psi}$ using a representative subset of the population. This ensures that the mixture remains a distinct global reference, maintaining the necessary contrast to identify policies that provide truly novel contributions to the state-space coverage.

4.2. Policy Compression

Following the generation of the policy dataset $\mathcal{D}_{\Theta}$ and the corresponding trajectory dataset $\mathcal{D}_{\tau}$, the framework proceeds to the Policy Compression phase. In this stage, we identify the generative mapping $g_{\zeta} : \mathcal{Z} \rightarrow \Theta$. While we could use many generative approaches to achieve this, we build upon the methodological framework introduced in [33] by utilizing an Autoencoder architecture. Specifically, for each high-dimensional parameter vector $\theta \in \mathcal{D}_{\Theta}$, the Autoencoder produces a reconstructed weight vector $\hat{\theta} = D_{\zeta}(E_{\xi}(\theta))$, where E_{ξ} and D_{ζ} denote the encoder and decoder, respectively.

The Autoencoder is trained by minimizing the Latent Behavior Compression objective (4), which we formulate as the Kullback-Leibler (KL) divergence between the mixture occupancy of the original dataset $d_{\mathcal{D}_{\Theta}}$ and the mixture occupancy induced by the reconstructed dataset $d_{\mathcal{D}_{\hat{\Theta}}}$, where $\mathcal{D}_{\hat{\Theta}} = \{D_{\zeta}(E_{\xi}(\theta_i))\}_{i=1}^M$.

Since an analytical description of these occupancy distributions is unavailable, we utilize the non-parametric importance sampling estimator described in Section 2.5 as our loss function. This allows us to evaluate the behavioral discrepancy using the particles in $\mathcal{D}_{\tau}$ without requiring new environment interactions. The loss function is then defined as:

$$\mathcal{L}_{AE}(\xi, \zeta) = D_{\text{KL}}(d_{\mathcal{D}_{\Theta}} \parallel d_{\mathcal{D}_{\hat{\Theta}}}) \approx \frac{1}{N} \sum_{i=1}^N \ln \frac{k/N}{\sum_{j \in \mathcal{N}_i^k} w_j}, \quad (9)$$

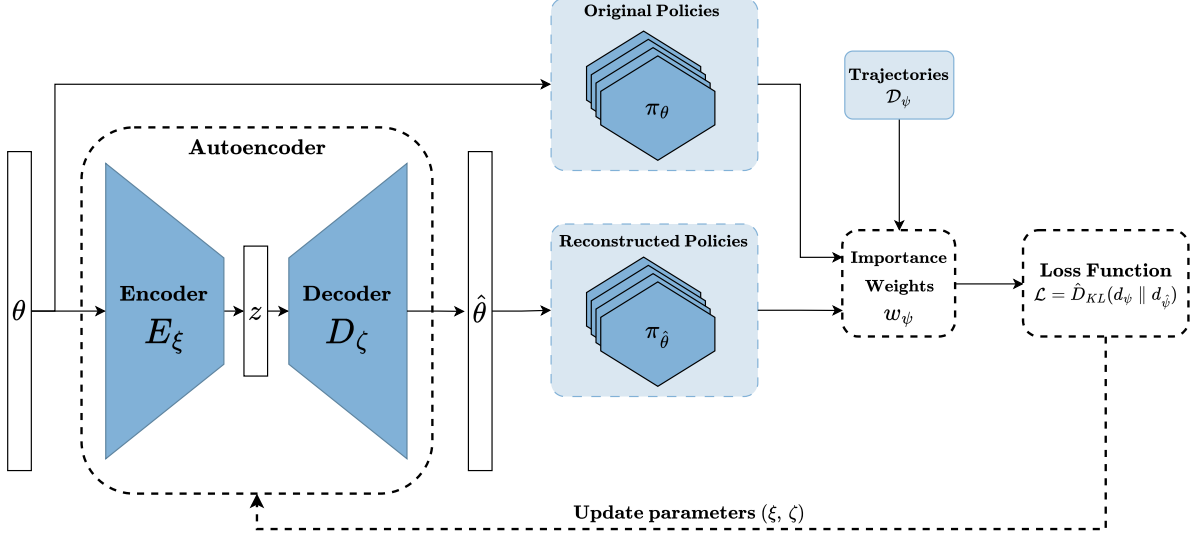


Figure 4: Autoencoder training pipeline. The weights of the policies from the mini-batch ψ are reconstructed by the autoencoder, and, using the relative trajectories $\mathcal{D}_\psi$, the importance sampling estimator approximates the discrepancy between the original and reconstructed mixture occupancy distributions to update the autoencoder parameters.

where N is the total number of particles in $\mathcal{D}_\tau$, $\mathcal{N}_i^k$ denotes the set of indices of the k -nearest neighbors of the i -th particle, and w_j are the self-normalized importance weights. Upon completion of the training, the decoder D_ζ is extracted to serve as the generative mapping g .

Importance Weights Calculation The non-normalized importance weight $\bar{w}_j$ associated with the j -th trajectory $\tau_j \in \mathcal{D}_\tau$ is defined as the ratio between the reconstructed (target) and dataset (behavioral) mixture probability density of observing that trajectory:

$$\bar{w}_j = \frac{\sum_{\hat{\theta} \in \mathcal{D}_{\hat{\Theta}}} \Pr(\tau_j | \pi_{\hat{\theta}})}{\sum_{\theta \in \mathcal{D}_{\Theta}} \Pr(\tau_j | \pi_{\theta})}, \quad (10)$$

where $\Pr(\tau_j | \pi_{\theta})$ is the probability of observing trajectory τ_j using policy π_{θ} . Since the policies share the same environment dynamics P and initial state distribution, and the mixtures have the same number of components, these terms cancel out, allowing the importance weights to be expanded into a policy-explicit form that is, effectively, what allows the loss to be differentiable:

$$\bar{w}_j = \frac{\sum_{\hat{\theta} \in \mathcal{D}_{\hat{\Theta}}} \prod_{t=0}^T \pi_{\hat{\theta}}(a_t^j | s_t^j)}{\sum_{\theta \in \mathcal{D}_{\Theta}} \prod_{t=0}^T \pi_{\theta}(a_t^j | s_t^j)}, \quad (11)$$

where $\tau_j = (s_0^j, a_0^j, \dots, s_T^j, a_T^j)$. The final importance weights are then obtained through self-normalization: $w_j = \bar{w}_j / \sum_{i=1}^N \bar{w}_i$, where N is the total number of trajectories in $\mathcal{D}_\tau$.

This computation follows the same methodology used in [23], adapted here to accommodate mixtures of policies. Similarly to the original work, we associate each particle s_j with the importance weight w_j calculated for the trajectory τ_j that produced it.

It is important to note that Equation (11) cannot be used *as is* for numerical computation. Because it involves the product of numerous density values lying in the interval $[0, 1]$, both the numerator and denominator often vanish to zero, leading to arithmetic underflow and a loss of numerical precision. To ensure stability, we compute the importance weights in log-space using the *log-sum-exp* (LSE) trick. The non-normalized weights are calculated as:

$$\bar{w}_j = \exp \left\{ \text{LSE}_{i \in \mathcal{D}_{\hat{\Theta}}} (\ell'_{i,j}) - \text{LSE}_{i \in \mathcal{D}_{\Theta}} (\ell_{i,j}) \right\}, \quad (12)$$

where $\ell_{i,j}$ and $\ell'_{i,j}$ denote the log-probabilities of trajectory τ_j under the i -th policy of the behavior and target mixtures, respectively. A detailed breakdown of this computation and the subsequent self-normalization procedure is provided in Appendix C.

Training Pipeline To manage the computational complexity associated with a large policy population $\mathcal{D}_\Theta$, we adopt a stochastic optimization approach that operates on mini-batches of policies. In each training step, a subset ψ of P policies is sampled from the dataset, and their corresponding trajectories $\mathcal{D}_\psi$ are retrieved from the dataset $\mathcal{D}_\tau$ obtained during the Policy Dataset Generation phase. Thanks to the importance sample estimator $\hat{D}_{\text{KL}}$, we can perform I successive gradient updates to the autoencoder parameters ϕ using the same trajectory data $\mathcal{D}_\psi$, increasing sample efficiency and reducing variance. Within this loop, the importance weights are recalculated at every step to account for the evolving parameters of the reconstructed policies ψ , using the numerically stable formulation described by (12). The training continues until the autoencoder has converged or until the population has been sufficiently sampled, ensuring that the resulting latent space $\mathcal{Z}$ provides a continuous and high-fidelity representation of the entire behavioral repertoire of the original dataset. The practical implementation of the Latent Behavior Compression phase is detailed in Algorithm 1 and graphically depicted in Figure 4.

Algorithm 1 Latent Behavior Compression

Input: Policy dataset $\mathcal{D}_\Theta$, trajectory dataset $\mathcal{D}_\tau$, inner iterations I , batch size P , nearest neighbors parameter k , learning rate α .

Output: Generative behavioral mapping g_ζ .

- 1: Randomly initialize autoencoder parameters $\phi = \{\xi, \zeta\}$
 - 2: **repeat**
 - 3: Sample a mini-batch of P policies $\psi = \{\theta_i\}_{i=1}^P \subset \mathcal{D}_\Theta$
 - 4: Retrieve corresponding trajectories $\mathcal{D}_\psi \subset \mathcal{D}_\tau$
 - 5: **for** $i = 1$ **to** I **do**
 - 6: Reconstruct parameters: $\hat{\psi} = \{\hat{\theta}_i = D_\zeta(E_\xi(\theta_i)) \mid \theta_i \in \psi\}$
 - 7: Compute mixture-matching loss: $\mathcal{L}(\phi) = \hat{D}_{\text{KL}}(d_\psi \parallel d_{\hat{\psi}})$ {via Eq. 9}
 - 8: Update weights: $\phi \leftarrow \phi - \alpha \nabla_\phi \mathcal{L}(\phi)$
 - 9: **end for**
 - 10: **until** convergence or full population coverage
 - 11: **return** D_ζ as g_ζ
-

4.3. Latent Behavior Optimization

Once the behavioral manifold is recovered through the unsupervised compression phase, the generative mapping $g_{\zeta^*} : \mathcal{Z} \rightarrow \Theta$ provides a compact parameterization of the agent’s reachable behavior repertoire.

At this stage, the unsupervised exploration is followed by a supervised learning phase: given a specific task defined by a reward signal R , the agent must identify the latent code $\mathbf{z}^* \in \mathcal{Z}$ that maximizes the expected return J^R . As illustrated in Figure 5, the integration of the frozen decoder D_{ζ^*} , meaning the parameter vector ζ^* is kept constant throughout the process, with the policy network π_θ creates a composite latent policy, defined as $\pi_{\mathbf{z}} = \pi_{g_{\zeta^*}(\mathbf{z})}$. From a theoretical standpoint, calculating the gradient of the performance objective with respect to the latent code would require the application of the chain rule:

$$\nabla_{\mathbf{z}} J^R(\mathbf{z}) = \nabla_{\mathbf{z}} g_{\zeta^*}(\mathbf{z})^\top \nabla_{\theta} J^R(\theta), \quad (13)$$

where $\nabla_{\theta} J^R(\theta)$ represents the conventional policy gradient and $\nabla_{\mathbf{z}} g_{\zeta^*}(\mathbf{z})$ denotes the Jacobian of the decoder.

However, rather than relying on direct back-propagation, we choose to optimize this latent policy utilizing *Policy Gradient with Parameter-based Exploration* (PGPE, [29]). We chose PGPE because, first and foremost, it enables the identification of

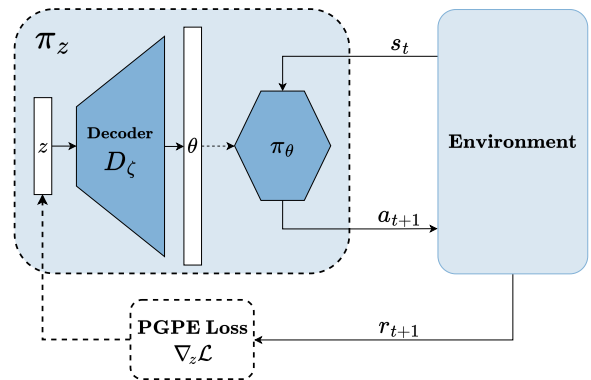


Figure 5: Latent Behavior Optimization. The latent code $\mathbf{z}$ is passed to the decoder D_ζ to generate the policy parameters θ . The composite system acts as a latent-parameterized policy $\pi_{\mathbf{z}}$. PGPE explores the latent manifold $\mathcal{Z}$ by updating the distribution over $\mathbf{z}$ based on the reward signal r from the environment.

the optimal latent code $\mathbf{z}$ without the need to compute the Jacobian of the decoder; furthermore, since the latent space $\mathcal{Z}$ is significantly more compact than the original parameter space ($k \ll n$), the reduced search space allows for a more dense and efficient exploration of the behavioral manifold. This efficiency is further enhanced by the smoothness of the latent space: because the compression phase was topologically structured by an occupancy-matching objective, proximity in $\mathcal{Z}$ directly correlates with behavioral similarity. This results in a notably more stable optimization landscape compared to the redundant and often chaotic nature of the raw parameter space Θ .

In this setup, we freeze the generative mapping parameters ζ^* and initialize a hyper-policy μ_ϕ , a distribution over the latent space parameterized by ϕ . The agent then iteratively interacts with the environment, leveraging the feedback from the reward signal to refine the hyperparameters ϕ until an optimal latent behavior $\mathbf{z}^* \in \mathcal{Z}$ is identified.

5. Experiments and results

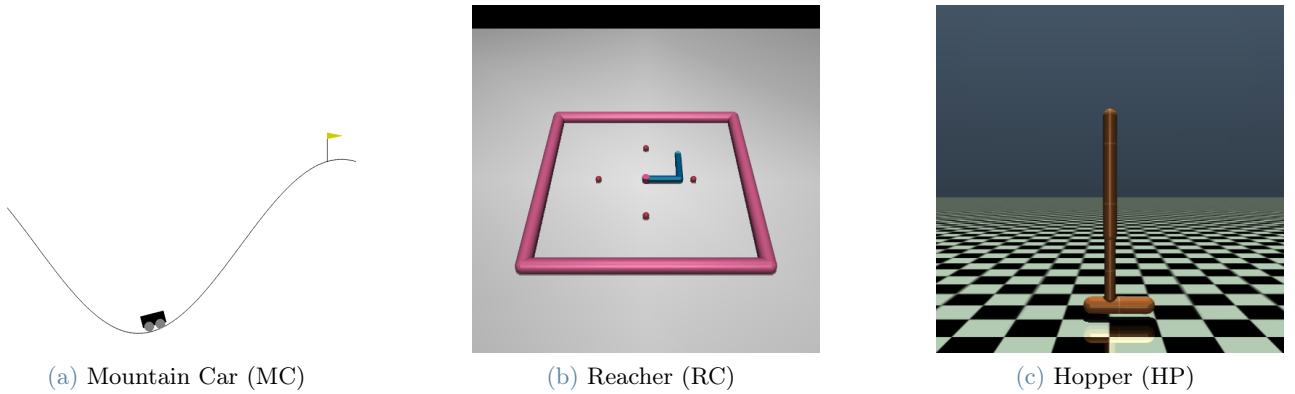


Figure 6: Experimental environments.

5.1. Experimental Setup

This section provides an empirical validation of the proposed framework, assessing its ability to identify and exploit behavioral manifolds in complex environments. The experiments are designed to verify the effectiveness of the three-stage pipeline detailed in Section 4 through the following evaluations:

1. **Dataset Quality and Filtering (Section 5.2):** We assess how the uniqueness score ω_i influences the distribution of behaviors within the dataset, specifically testing whether it improves the balance between common and rare high-performing behaviors.
2. **Manifold Topology (Section 5.3):** We investigate the geometric properties of the learned latent space $\mathcal{Z}$ by testing for behavioral continuity. We further evaluate how the uniqueness metric leads to a more informative manifold compared to random policy selection.
3. **Optimization Efficiency (Section 5.4):** We measure the sample efficiency of latent behavior optimization, comparing its convergence speed and final performance against standard reinforcement learning baselines.
4. **Architectural Robustness (Section 5.4):** We evaluate the framework’s effectiveness across varying policy network scales.

To test the framework and perform the experiments, we consider a classical control problem and two higher-dimensional robotic simulations. To test the versatility of the learned manifolds, we also consider four different tasks for each environment.

The first environment is **Mountain Car Continuous** (MC, [21]), which, despite its low-dimensional state space, requires significant exploration to reach specific goal states. We define four downstream tasks: **standard** and **left**, which require reaching opposite peaks; and **speed** and **height**, which reward the agent for maintaining high kinetic and potential energy, respectively, without terminating the episode.

To evaluate the framework in more complex settings, we utilize two environments from the MuJoCo suite [32, 34]:

- **Reacher (RC):** A two-link robotic arm. Tasks include **speed** (linear fingertip velocity), **clockwise** and **counter-clockwise** rotation, and **radial** speed movement (alternating extension and retraction).
- **Hopper (HP):** A planar monopod that rewards survivability. The rewards mainly focus on locomotion

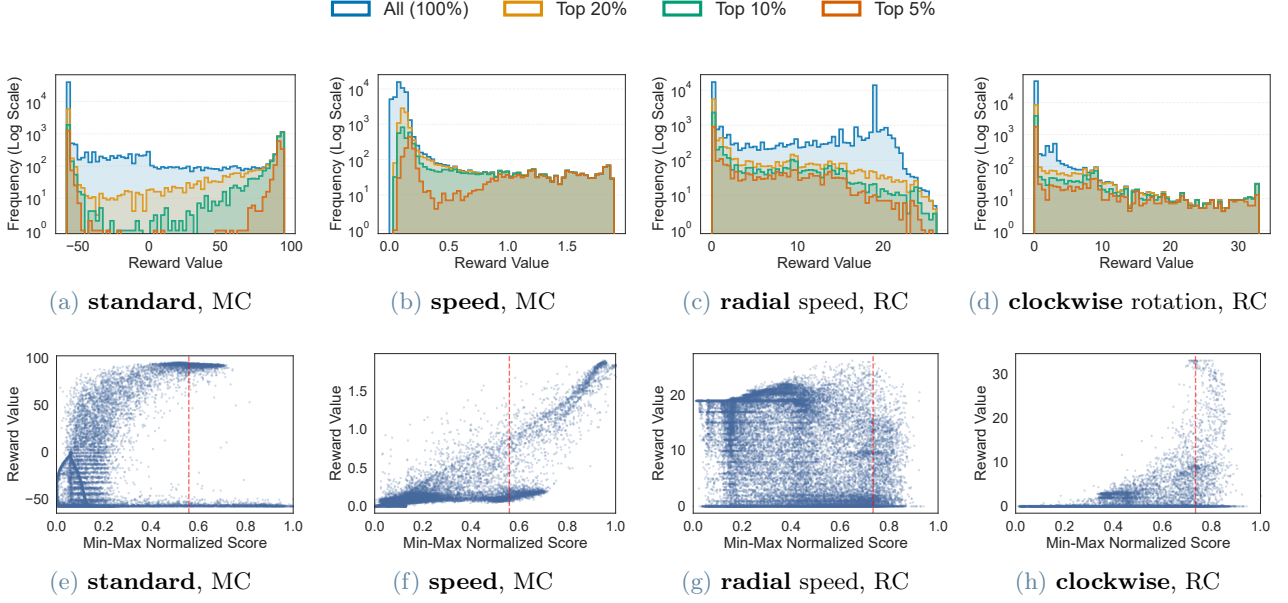


Figure 7: Effect of score filtering on the population for the MC and RC environments. The top row illustrates the log-frequency of rewards across different selection percentiles, and the bottom row shows the correlation between the uniqueness scores and task-specific rewards, with the vertical dashed line indicating the top 5% selection threshold.

(**forward**, **backward**, and **standstill**). We also considered the **standard** reward as defined by the Gymnasium Documentation: $healthy + forward - control$, where *healthy* is a constant reward for each timestep the monopod is in a healthy state (upright and tall), *forward* is a reward proportional to the speed in the right direction, and *control* is a negative reward that penalizes large actions.

We consider three policy sizes: Small, Medium, and Large. Unless specified otherwise, all experiments start with a dataset of $M = 50,000$ initial policies of Medium size. Additional details regarding the network architectures and hyperparameters, as well as exhaustive documentation on the environments and rewards, are presented in Appendix D.

5.2. Policy Dataset Generation

The first stage of our experimental campaign evaluates the effectiveness of the uniqueness metric ω_i in curating a high-quality policy dataset. We hypothesize that by maximizing behavioral uniqueness, we can significantly reduce the redundancy of a randomly initialized population while preserving the optimal behaviors present in the dataset required for downstream task success.

To test this, we generate a population of random policies and analyze how the distribution of task-specific rewards evolves as we apply increasingly stringent filtering based on ω_i . Figure 7 illustrates this process for the Mountain Car (MC) and Reacher (RC) environment across different reward functions.

As demonstrated in the top row of Figure 7, the uniqueness score allows for an aggressive compression of the dataset, retaining as little as 5% of the original population, without sacrificing the presence of high-performing behaviors. While the total population is dominated by low-reward behaviors, the sub-populations isolated by higher scores exhibit a marked reduction in mass toward the left-hand tail of the reward distribution.

This phenomenon is further shown by the scatter plots in the bottom row (Figures 7e to 7h). Despite being computed in a completely task-agnostic manner, the uniqueness scores exhibit a positive correlation with all four downstream rewards. This suggests that the information-theoretic criteria of individual state-occupancy entropy and divergence from the mixture act as effective proxies for functional behaviors. Even for rewards that don't correlate with the scores as well as others, such as **standard** for MC and **radial speed** for RC, it's still possible to retain most of the optimal policies by performing a less aggressive filtering as shown in Figures 7a and 7c. Additional reward distributions across all environments and policy sizes are shown and analyzed in Appendix E.1.

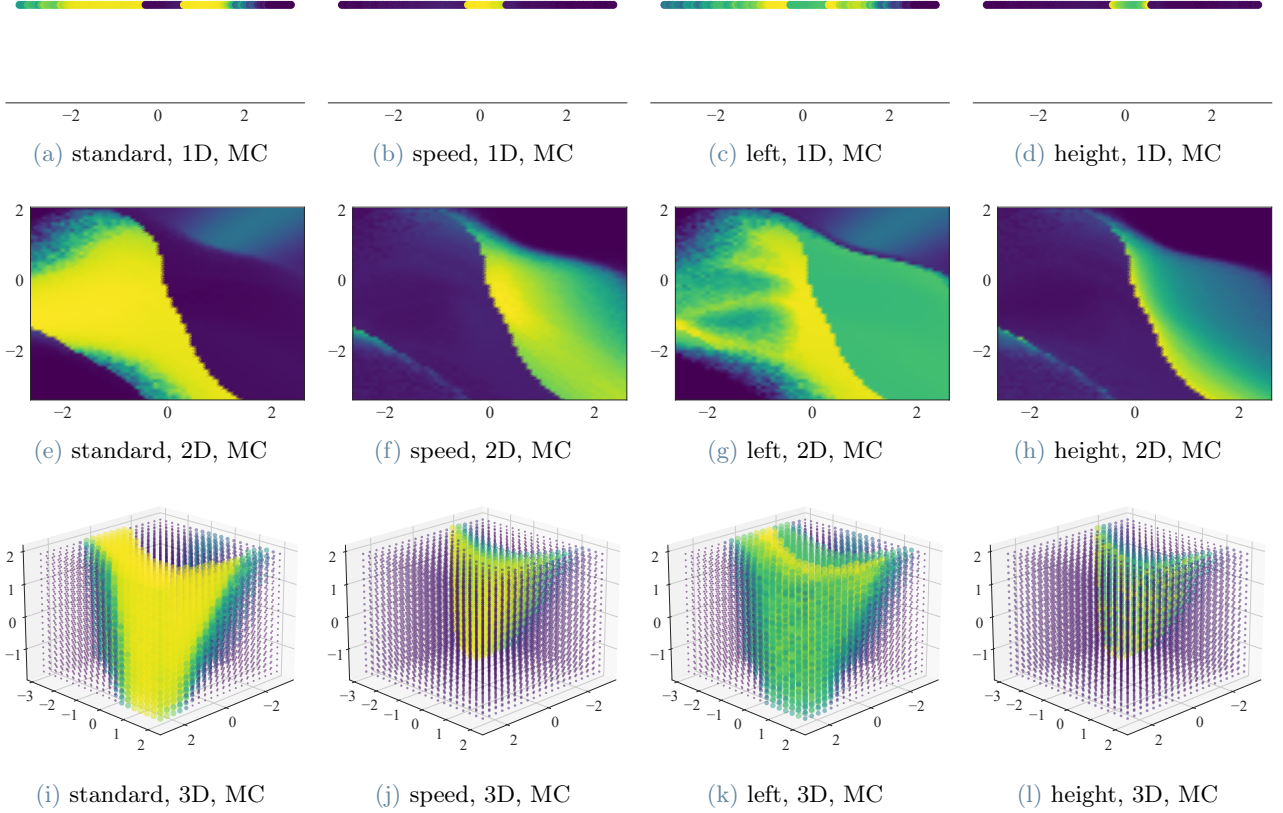


Figure 8: Visualization of the behavioral manifolds across different latent dimensions and rewards for the MC environment. **Lighter** colors indicate higher rewards, while **darker** colors indicate lower rewards.

5.3. Latent Behavior Compression

Following the unsupervised compression phase, we evaluate the quality and geometric regularity of the resulting latent spaces. We use this analysis mainly to validate the *Manifold Hypothesis* within our framework: for the latent space $\mathcal{Z}$ to serve as an effective search space, it must exhibit *behavioral continuity*, meaning that we expect that proximal points in the latent manifold map to policies that induce similar state-occupancy distributions and, consequently, similar reward values.

To interrogate the latent geometry, we consider a subspace extending slightly beyond the regions identified by the training policies. We then define a uniform grid of latent codes and decode them into policy parameters via the learned mapping. Each generated policy is then evaluated in the environment to produce a performance map. Figure 8 illustrates the behavioral manifolds for latent spaces of 1, 2, and 3 dimensions, evaluated across various reward functions in the Mountain Car (MC) environment. A more comprehensive display of latent spaces is shown in Appendix E.2.

The empirical results confirm that the learned manifolds are fairly smooth and connected. As can be observed in the 2D heatmaps and 3D plots, high-reward policies are not isolated points but form expansive, contiguous areas within the latent space. This regularity suggests that the mixture-matching objective effectively aligns the latent geometry with the environment’s underlying dynamics. Furthermore, although the original policy networks reside in a redundant, high-dimensional parameter space, their functional diversity is successfully captured within a relatively compact subset of $\mathbb{R}^k$, spanning between small intervals centered around the origin. Another important characteristic that the performance maps reveal is that the boundaries of the reward regions may exhibit abrupt transitions depending on the task objective. For instance, in Subfigure 8f, the region of high **speed** reward terminates abruptly where behaviors that maximize the **standard** reward begin; a similar phenomenon is observed for the **height** reward. This behavior can be explained by the environment’s termination conditions and how they interact with the rewards: policies that achieve maximum speed or height are often those closest to reaching the goal state, which triggers an immediate end to the episode and prevents gaining additional reward. While it is theoretically possible to further maximize these signals by hovering near the goal without crossing it, such highly specific behaviors are unlikely to be generated via random initialization

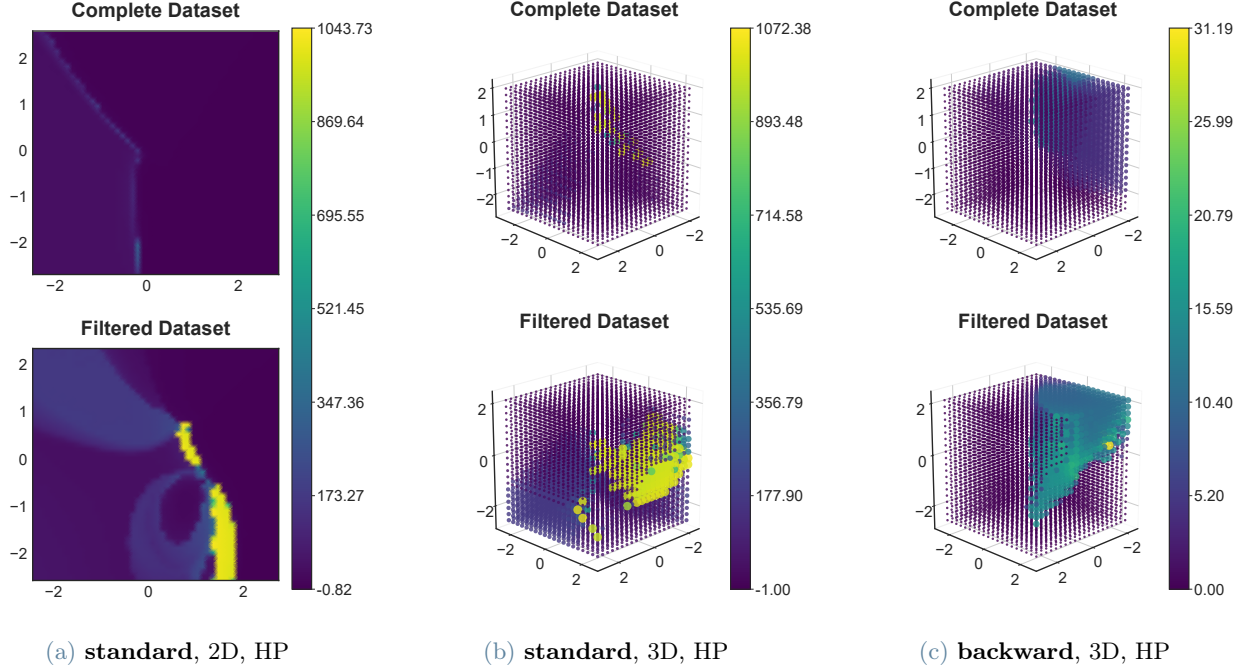


Figure 9: Comparison of behavioral manifolds in the HP environment. Each subfigure contrasts the latent space derived from a randomly downsampled dataset (top) against the one obtained by selecting the top 5% policies according to their scores (bottom).

and are therefore not present in the unsupervised training set. This shows that the autoencoder is only able to reconstruct the behavioral repertoire it was exposed to.

Contribution of the Uniqueness Scores The impact of the score-based filtering is particularly evident when comparing the resulting latent topographies. To have a fair comparison, the unfiltered dataset was uniformly downsampled to match the cardinality of the filtered dataset, and the initial training conditions were kept the same. We use as examples of this the latent spaces evaluated on the **standard** and **backward** reward for the HP environments (Figure 9).

By comparing the topographies, it is clear that the autoencoders trained on unfiltered data have allocated the vast majority of areas to the most common low-value policies. We hypothesize that, due to the statistical over-representation of failure-mode behaviors, such as policies that quickly end the episode by falling, the autoencoder must allocate its limited bottleneck capacity to represent these low-utility regions to match the target mixture occupancy. In contrast, by intentionally selecting the top 5% of behaviorally unique policies, the compression process is effectively denoised by the removal of useless behaviors. This selection expands the areas of high-performing policies, providing a significantly more valuable landscape for downstream task optimization.

5.4. Latent Optimization

The final stage of our evaluation focuses on the practical utility of the learned manifold for reward maximization tasks. We hypothesize that by restricting the search space to a low-dimensional, behaviorally-dense manifold $\mathcal{Z}$, we can achieve significant gains in sample efficiency and optimization stability compared to standard DRL algorithms that work on the high-dimensional parameter space Θ .

To test this hypothesis, we utilize *Policy Gradient with Parameter-based Exploration* (PGPE, [29]) to optimize the latent codes, referred to as Latent PGPE. We compare this approach against several state-of-the-art DRL baselines operating in the full parameter space Θ : PPO [28], DDPG [18], TD3 [7], and SAC [10]. Additionally, we include Parameter PGPE (operating directly on Θ) as a control to isolate the benefits of manifold compression from the benefits of parameter-based exploration.

For these experiments, the manifolds were derived from populations curated via the uniqueness score: for the Mountain Car (MC) and Hopper (HP) environments, we utilized the top 5% filtered datasets, while for the Reacher (RC) environment, the top 10% was employed. To ensure statistical robustness, we consider the top 10 seeds out of 15 runs against the baselines for each reward and environment.

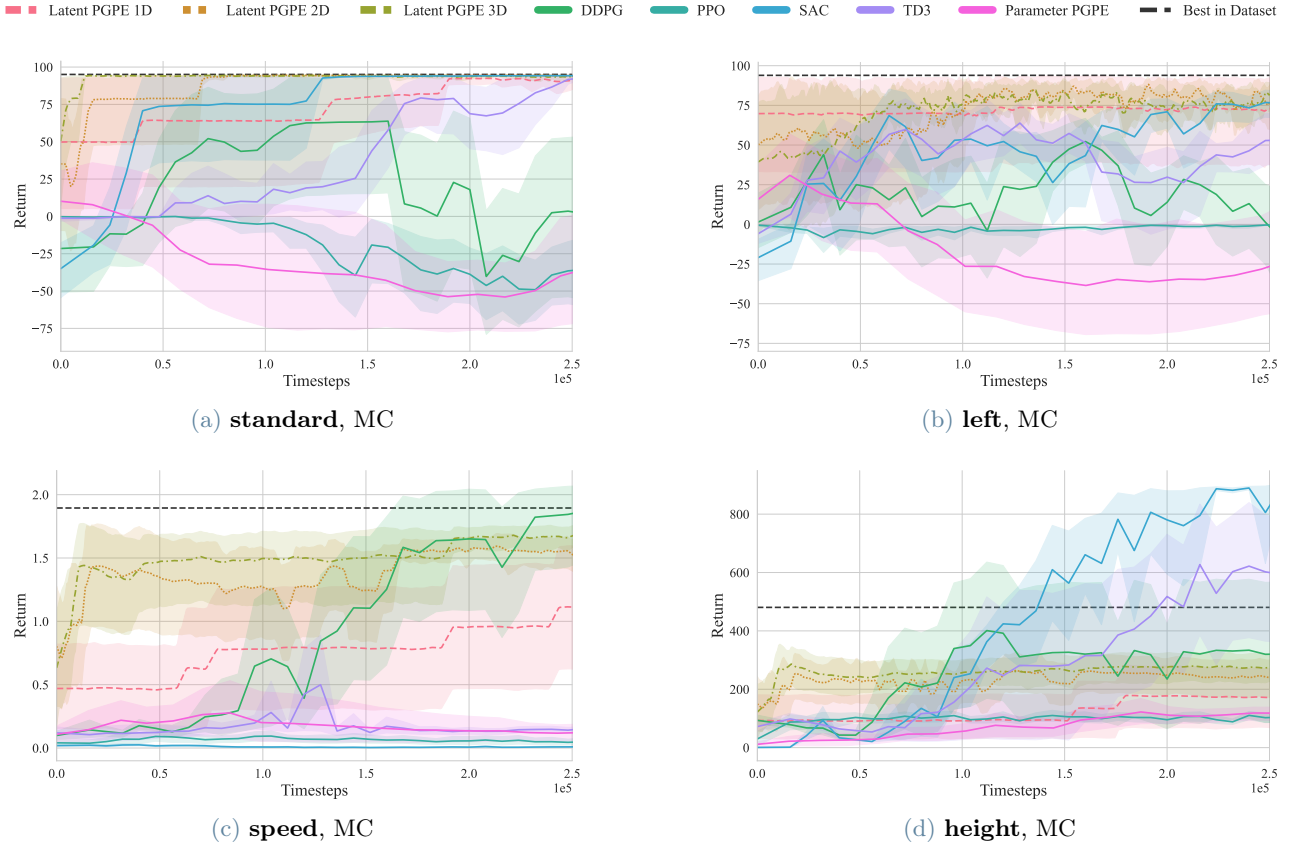


Figure 10: Performance comparison in MC for different tasks. We report the average and 95% confidence interval over the 10 best runs out of 15 and the reward of the best performing policy in the training dataset.

Performance and Sample Efficiency Across all environments, the empirical results validate that restricting the search to the behavioral manifold yields a significant increase in sample efficiency. As illustrated in Figures 10, 11, and 12, Latent PGPE matches or exceeds the convergence speed and performance of all baselines in 6 out of 12 tasks. Notably, the framework identifies functional, high-performing policies within the first few thousand interactions, whereas standard baselines often require significantly more data to exit the initial low-reward regime.

While standard methods suffer from a *cold-start* problem, navigating a chaotic Θ with no prior knowledge of environmental dynamics, Latent PGPE starts with a structured repertoire of valid behaviors that allow it to get to the best ones quicker. We also observe that Parameter PGPE consistently struggles with sample efficiency and exploration; in MC, it fails to find goal states for sparse rewards (Figures 10a and 10b), and in HP, it exhibits sluggish convergence in three out of four tasks. This shows that the success of the framework is due to the manifold topology rather than the optimization algorithm alone.

The Behavioral Ceiling and Latent Smoothing A recurring observation across the performance plots is that Latent PGPE sometimes fails to reach the absolute maximum reward obtained by the best baselines, or even the maximum reward present in the original training dataset.

In the MC environment, for instance, the **speed** and **height** tasks (Figure 10c, 10d) show a gap in asymptotic performance between Latent PGPE and some baselines. As discussed in Section 5.3, maximizing these rewards requires hovering precisely near the episode-ending goal without triggering termination, a very niche behavior that is statistically rare in a random population. If only a handful of policies out of the 2500 considered exhibit a specific behavior, the Autoencoder’s mixture-matching objective will naturally prioritize the reconstruction of more common behavioral modes to minimize the global divergence. This results in a latent *smoothing effect*: the manifold represents the most consistent and representative strategies, but may lose the ability to reconstruct extreme outliers. This phenomenon is further discussed in Appendix E.1 for the HP environment. Similarly, across all tasks, Latent PGPE never surpasses the best policy in the training set, indicating that the starting repertoire constitutes a hard *behavioral ceiling* that the autoencoder is not able to exceed through generalization.

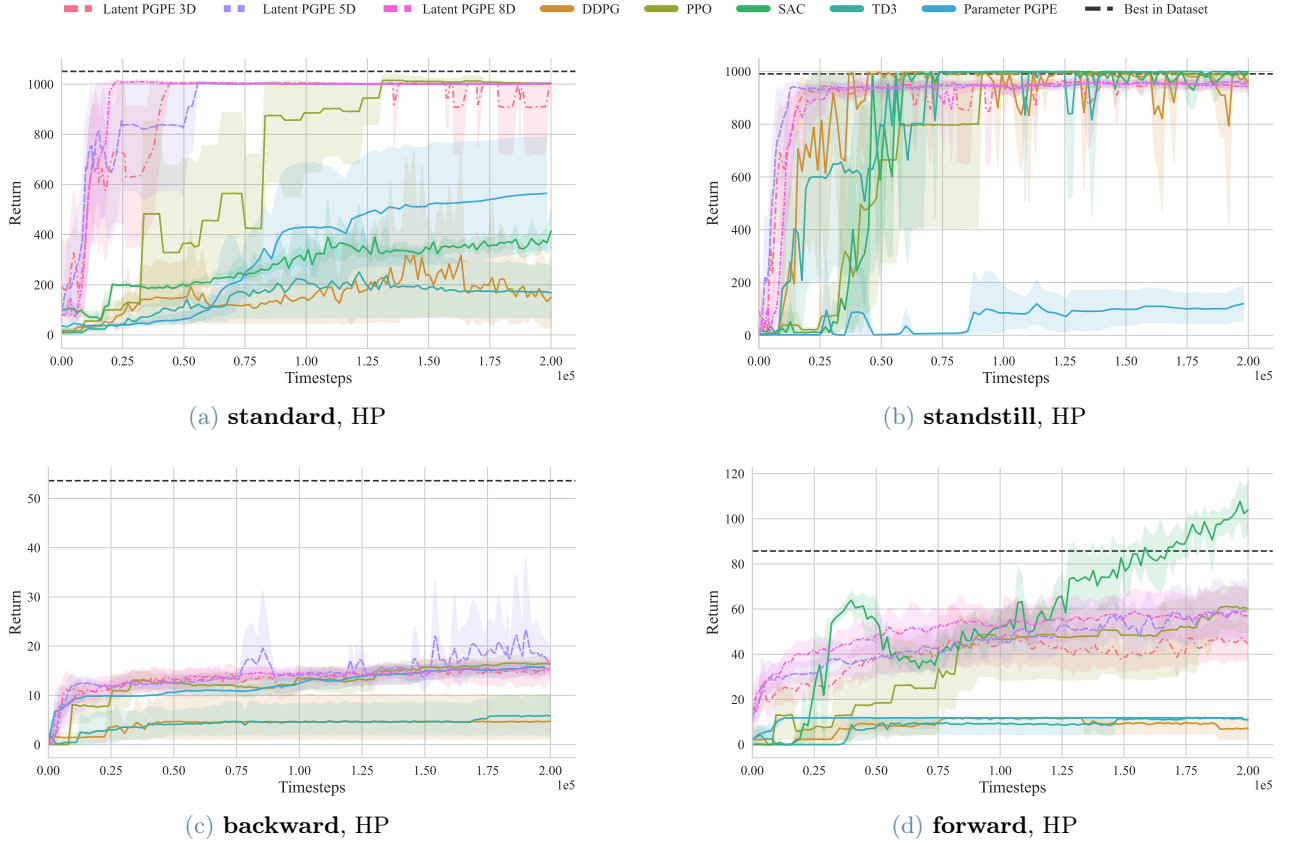


Figure 11: Performance comparison in HP for different tasks. We report the average and 95% confidence interval over the 10 best runs out of 15 and the reward of the best performing policy in the training dataset.

Analysis of MuJoCo Environments (HP and RC) For the Hopper environment (Figure 11), the framework exhibits notable robustness. All tested latent configurations yield similar convergence rates and performances that match or exceed those of the majority of baselines, trailing only SAC in the **forward** task (Figure 11d). This suggests that locomotion-based high-reward behaviors are effectively captured even under significant compression, despite the underlying higher-dimensional state space.

In the Reacher environment, Latent PGPE exhibits rapid convergence for the **speed** and **clockwise** rotation tasks (Figures 12a, 12c), outperforming baselines in the early stages of training, while settling for slightly lower final rewards. Conversely, performance in the **radial speed** task (Figure 12b) falls significantly short of the baselines. As shown in the reward distributions of Figure 7c, the population is devoid of policies capable of radial movement that yield rewards similar to those reached by the baselines. While random initialization easily produces directional biases such as rotation, it rarely yields the coordination required for radial extension and retraction. Consequently, the manifold’s asymptotic performance is strictly constrained by the behavioral support of the training dataset. This shows that the framework cannot optimize for behaviors that were never discovered during the initial unsupervised phase, indicating that our Latent Compression method is incapable of generalizing (and generating) new, unseen behaviors.

Impact of Latent Dimensionality We observe a consistent positive correlation between the dimensionality of the latent space and optimization performance. Higher-dimensional latent spaces provide more degrees of freedom, allowing for the representation of more nuanced control signals and a finer partitioning of the behavioral manifold.

While low-dimensional manifolds ($k \leq 3$) are sufficient for simpler tasks, we found that for the complex physics of MuJoCo environments (RC and HP), larger bottlenecks were required. Specifically, increasing the latent dimensionality to $k = 8$ was necessary to compete with the performance of state-of-the-art baselines. This suggests an *elbow* in the compression-utility trade-off since, although the manifold is low-dimensional compared to Θ , there is a critical threshold of latent variables required to maintain the functional diversity necessary for representing higher-precision control. Above this threshold, PGPE can navigate the space more effectively, whereas below it, the over-compression of behaviors impairs the agent’s ability to fine-tune its strategy.

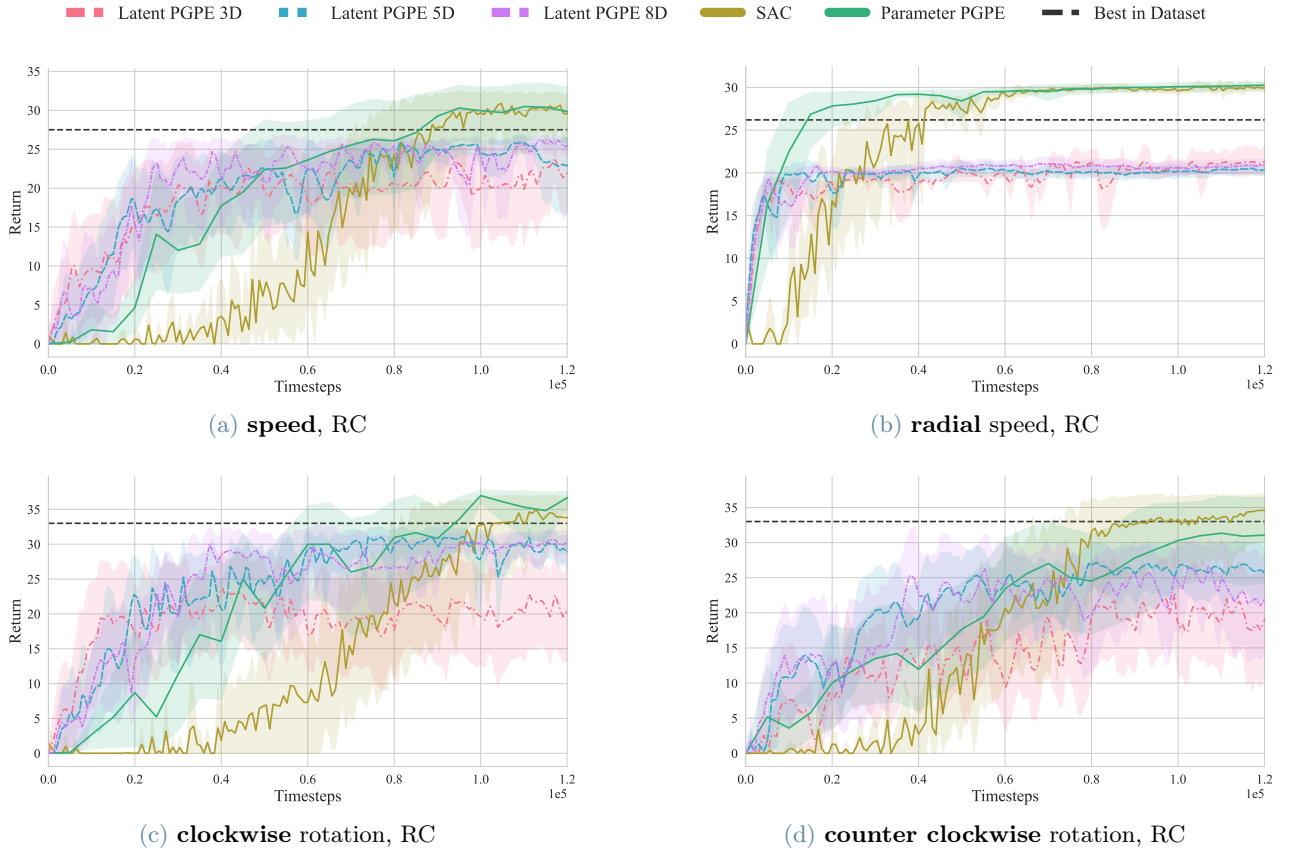


Figure 12: Performance comparison in RC for different tasks. We report the average and 95% confidence interval over the 10 best runs out of 15 and the reward of the best performing policy in the training dataset. For clarity, the worst-performing baselines are omitted.

Impact of Policy Architecture Finally, to verify the robustness of the framework, we evaluated the Latent Optimization phase across three different policy architectures: **Small**, **Medium**, and **Large** (detailed in Appendix D). Figure 13 illustrates the performance curves for these architectures in the MC environment. The results indicate that the framework successfully identifies and exploits a behavioral manifold across all considered parameter scales. However, we observe that the quality of the manifold is positively correlated with the network’s capacity. In the **speed** and **height** tasks (Figures 13c, 13d), the **Large** architecture achieves higher asymptotic rewards compared to its smaller counterparts. This suggests that the increased expressivity of larger networks allows the decoder to map latent codes to more nuanced behaviors, which are required for high-precision tasks.

Furthermore, in the **standard** task (Figure 13a), while all sizes eventually reach the goal, the **Medium** and **Large** networks converge faster than the **Small** network. Collectively, these findings suggest that while the behavioral manifold is an intrinsic property of the environment, its practical utility is enhanced by the increased capacity of the policy representation. These hypotheses are further corroborated by the results displayed in Appendix E.1, in which it’s evident from the reward distributions that datasets of higher-capacity policies don’t include significantly more high-value behaviors and are not, therefore, advantaged by a higher representation of task-optimal training behaviors.

6. Conclusions

This thesis has explored the hypothesis that the space of realizable behaviors in Deep Reinforcement Learning is intrinsically low-dimensional. In particular, we examined a novel approach to Unsupervised Reinforcement Learning that decouples behavioral discovery from task-specific optimization. Rather than operating within conventional high-dimensional parameter spaces, the proposed framework performs optimization directly on structured behavioral manifolds, enabling a more compact and meaningful representation of agent behaviors. Our experimental results across classic control and MuJoCo benchmarks confirm that the Manifold Hypothesis holds in the context of DRL. The identification of a latent space $\mathcal{Z}$ through an occupancy-matching objective has yielded three primary insights:

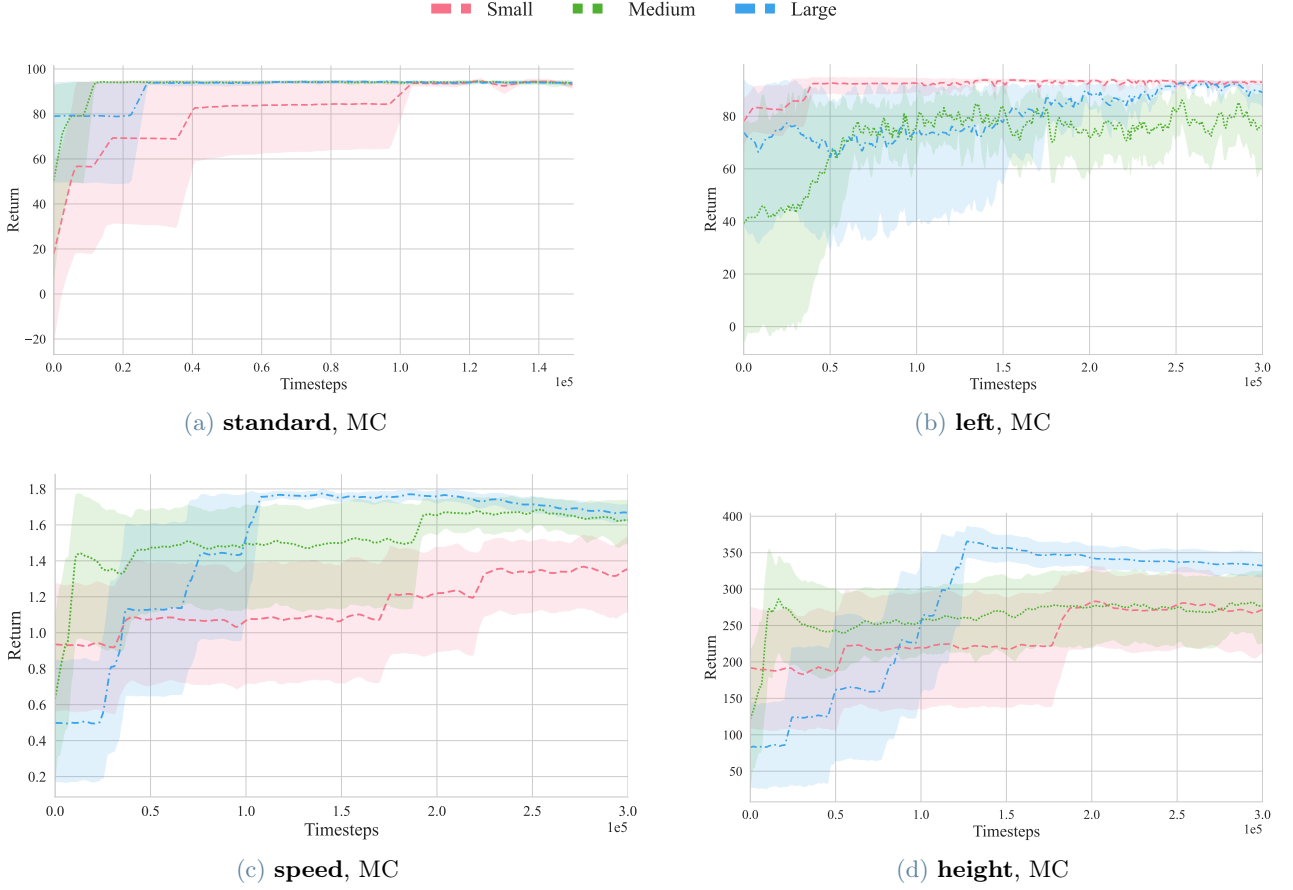


Figure 13: Performance comparison between different policy sizes in MC. We report the average and 95% confidence interval over the 10 best runs out of 15 and the reward of the best performing policy in the training dataset.

1. **Sample Efficiency:** Optimization in the latent space can significantly outperform standard Policy Gradient methods in terms of convergence speed, often identifying functional strategies within a fraction of the interactions required by baselines.
2. **Topological Regularity:** The mixture-matching objective effectively structures the latent manifold such that behavioral similarity is directly reflected in its geometric structure. This regularity provides a stable landscape for optimization, mitigating the *cold-start* problem typical of Policy Gradient methods operating in high-dimensional neural networks.
3. **Architectural Robustness:** The framework identifies functional latent representations across policy architectures of varying sizes, hinting that the behavioral manifold represents an inherent property of the environment-agent interaction. We also observed that latent optimization performance correlates with network capacity. This indicates that while behavioral compression is universal, its fidelity is linked to the expressivity of the underlying parametric model.

However, the framework also exposed a fundamental limitation, termed the **Behavioral Ceiling**. This phenomenon refers to the upper bound on asymptotic performance observed during the final latent optimization stage, which stems from an insufficient representation of optimal behaviors in the dataset used for behavioral compression. The root cause can be traced to a **Latent Smoothing** effect, which, although beneficial for denoising the parameter space, acts as a double-edged sword by favoring the reconstruction of dominant behavioral modes and consequently attenuating niche or infrequent behaviors present in the original dataset. As a result, even when score-based filtering is applied, the achievable performance remains ultimately constrained by the diversity captured during the initial unsupervised behavioral discovery phase and the fraction of optimal policies within the population.

6.1. Future Works

While this work provides an exploratory view for latent behavior compression and optimization, several avenues remain for future research to address its current limitations.

Active Behavioral Discovery The current pipeline relies on a large population of randomly initialized policies, filtered through an information-theoretic uniqueness metric. While effective for simple motor skills, random initialization is a passive discovery method that struggles to uncover complex, multi-stage, or temporally extended behaviors (e.g., the radial movement in Reacher or precise balancing in Hopper). Future work should integrate **active task-agnostic exploration** methods, such as Maximum Entropy exploration or intrinsic motivation frameworks, to generate the initial dataset. By replacing thousands of random policies with a smaller set of specialized policies, the framework could focus its capacity on a more diverse and functional behavioral repertoire.

Trajectory Density and Divergence Estimation The non-parametric KL-divergence estimator used in the compression phase is highly sensitive to the number of trajectories per policy. A potential improvement lies in shifting from a *wide* dataset (many policies, few trajectories) to a *deep* dataset (fewer, highly unique policies with many trajectories). This would allow for a much finer estimation of state-occupancy distributions, potentially reducing the smoothing effect and allowing the autoencoder to capture high-performing outliers without being overwhelmed by failure-mode noise.

Alternative Latent Optimizers While PGPE was chosen for its derivative-free nature and efficiency in low-dimensional spaces, it may not be the optimal tool for all manifolds as PGPE assumes a certain degree of unimodality in the search space. Future research could explore *Latent-PPO* [11] to leverage the gradients of the policy itself, or even *Bayesian Optimization* in the latent space to handle potentially multi-modal reward landscapes.

References

- [1] Jiří Ajgl and Miroslav Šimandl. Differential entropy estimation by particles. *IFAC Proceedings Volumes*, 44(1):11991–11996, 2011.
- [2] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. Pmlr, 2017.
- [3] Lawrence Cayton et al. Algorithms for manifold learning. *Univ. of California at San Diego Tech. Rep*, 12(1-17):1, 2005.
- [4] Marc Peter Deisenroth, Gerhard Neumann, and Jan Peters. A Survey on Policy Search for Robotics. *Foundations and Trends in Robotics*, 2(1-2):1–142, August 2013.
- [5] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. In *International Conference on Learning Representations*, 2018.
- [6] Charles Fefferman, Sanjoy Mitter, and Hariharan Narayanan. Testing the manifold hypothesis. *Journal of the American Mathematical Society*, 29(4):983–1049, 2016.
- [7] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018.
- [8] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.
- [9] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. *Advances in neural information processing systems*, 30, 2017.
- [10] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. Pmlr, 2018.
- [11] Xiang Han. LG-H-PPO: offline hierarchical PPO for robot path planning on a latent graph. *Frontiers in Robotics and AI*, 12, January 2026.
- [12] G. E. Hinton and R. R. Salakhutdinov. Reducing the Dimensionality of Data with Neural Networks. *Science*, 313(5786):504–507, 2006.

- [13] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, and others. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, 29(6):82–97, 2012.
- [14] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. *Advances in neural information processing systems*, 29, 2016.
- [15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012.
- [16] Michael Laskin, Denis Yarats, Hao Liu, Kimin Lee, Albert Zhan, Kevin Lu, Catherine Cang, Lerrel Pinto, and Pieter Abbeel. Urlb: Unsupervised reinforcement learning benchmark. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.
- [17] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.
- [18] Timothy Paul Lillicrap, Jonathan James Hunt, Alexander Pritzel, Nicolas Manfred Otto Heess, Tom Erez, Yuval Tassa, David Silver, and Daniel Pieter Wierstra. Continuous control with deep reinforcement learning, sep 15 2020. US Patent 10,776,692.
- [19] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space, September 2013.
- [20] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning, December 2013.
- [21] Andrew William Moore. Efficient memory-based learning for robot control. Technical Report UCAM-CL-TR-209, University of Cambridge, Computer Laboratory, 1990.
- [22] Mirco Mutti, Stefano Del Col, and Marcello Restelli. Reward-Free Policy Space Compression for Reinforcement Learning, 2022.
- [23] Mirco Mutti, Lorenzo Pratissoli, and Marcello Restelli. Task-Agnostic Exploration via Policy Gradient of a Non-Parametric State Entropy Estimate. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(10):9028–9036, May 2021.
- [24] Saeid Nahavandi, Roohallah Alizadehsani, Darius Nahavandi, Shady Mohamed, Navid Mohajer, Mohammad Rokonzaman, and Ibrahim Hossain. A comprehensive review on autonomous navigation. *ACM Computing Surveys*, 57(9):1–67, 2025.
- [25] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, August 2014.
- [26] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.
- [27] Andrei A. Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy Distillation, January 2016.
- [28] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
- [29] Frank Sehnke, Christian Osendorfer, Thomas Rückstieß, Alex Graves, Jan Peters, and Jürgen Schmidhuber. Parameter-exploring policy gradients. *Neural Networks*, 23(4):551–559, May 2010.
- [30] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dhharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm, December 2017.

- [31] R.S. Sutton and A.G. Barto. *Reinforcement Learning, second edition: An Introduction*. Adaptive Computation and Machine Learning series. MIT Press, 2018.
- [32] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy Lillicrap, and Martin Riedmiller. DeepMind Control Suite. *arXiv preprint arXiv:1801.00690*, January 2018.
- [33] Davide Tenedini, Riccardo Zamboni, Mirco Mutti, and Marcello Restelli. From Parameters to Behavior: Unsupervised Compression of the Policy Space. In *14th International conference on learning representations*, 2026.
- [34] Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, October 2012.
- [35] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.
- [36] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, May 1992.
- [37] Ming Zhang, Yawei Wang, Xiaoteng Ma, Li Xia, Jun Yang, Zhiheng Li, and Xiu Li. Wasserstein Distance guided Adversarial Imitation Learning with Reward Shape Exploration. In *2020 IEEE 9th Data Driven Control and Learning Systems Conference (DDCLS)*, pages 1165–1170, November 2020.

A. Derivation of mixture entropy formula

Let $M = |\Psi|$ denote the number of policies, and, for notation’s simplicity, let $d_i(s)$ for $i = 1, \dots, M$ be the state occupancy distribution induced by the i -th policy $\pi_i \in \Psi$, defined as the time-averaged probability of visiting state s .

We consider the uniform mixture over these occupancy distributions by assuming that each policy is selected with equal probability,

$$\Pr(\pi = \pi_i) = p(i) = \frac{1}{M}.$$

This induces a joint distribution over states and policies given by

$$p(s, i) = p(s | i) p(i) = \frac{1}{M} d_i(s), \quad (14)$$

from which the marginal state distribution, corresponding to the mixture occupancy d_Ψ , is obtained as

$$d_\Psi(s) = \sum_{i=1}^M p(s, i) = \sum_{i=1}^M d_i(s) p(i) = \frac{1}{M} \sum_{i=1}^M d_i(s).$$

Let $H(X)$ denote the differential entropy of a random variable X . Starting from the identity

$$H(X, Y) = H(X) + H(Y | X),$$

and using the symmetry of the joint differential entropy $H(X, Y) = H(Y, X)$, we obtain

$$\begin{aligned} H(X) + H(Y | X) &= H(Y) + H(X | Y) \\ \implies H(X) &= H(X | Y) + [H(Y) - H(Y | X)] \\ \implies H(X) &= H(X | Y) + MI(X, Y), \end{aligned} \quad (15)$$

where $MI(X, Y)$ denotes the mutual information between X and Y . In general, for a pair of random variables (X, Y) with joint density $p(x, y)$ and marginals $p(x)$ and $p(y)$, the mutual information is defined as

$$MI(X, Y) = \int_{\mathcal{X} \times \mathcal{Y}} p(x, y) \log \frac{p(x, y)}{p(x)p(y)} dx dy.$$

We now apply (15) to the random variables $S \sim d_\Psi(s)$ and $\pi \sim p(i)$, whose joint density is $p(s, i)$ as defined in (14). The conditional entropy becomes

$$\begin{aligned}
H(S \mid \pi) &= - \int_{S \times \{1, \dots, M\}} p(s, i) \log p(s \mid i) ds \\
&= - \int_S \sum_{i=1}^M p(s, i) \log p(s \mid i) ds \\
&= - \int_S \sum_{i=1}^M \frac{1}{M} d_i(s) \log d_i(s) ds \\
&= \frac{1}{M} \sum_{i=1}^M \left(- \int_S d_i(s) \log d_i(s) ds \right) \\
&= \frac{1}{M} \sum_{i=1}^M H(d_i).
\end{aligned}$$

The mutual information can be written as

$$\begin{aligned}
MI(S, \pi) &= \int_{S \times \{1, \dots, M\}} p(s, i) \log \frac{p(s, i)}{d_\Psi(s)p(i)} ds \\
&= \int_S \sum_{i=1}^M p(s, i) \log \frac{p(s, i)}{d_\Psi(s)p(i)} ds \\
&= \int_S \sum_{i=1}^M \frac{1}{M} d_i(s) \log \frac{\frac{1}{M} d_i(s)}{d_\Psi(s) \cdot \frac{1}{M}} ds \\
&= \frac{1}{M} \sum_{i=1}^M \int_S d_i(s) \log \frac{d_i(s)}{d_\Psi(s)} ds \\
&= \frac{1}{M} \sum_{i=1}^M D_{\text{KL}}(d_i \parallel d_\Psi).
\end{aligned}$$

Combining the two terms and writing the entropy as a functional of the density, we finally obtain the entropy decomposition of the mixture occupancy distribution:

$$H(d_\Psi) = \frac{1}{M} \sum_{i=1}^M [H(d_i) + D_{\text{KL}}(d_i \parallel d_\Psi)].$$

B. Upper Bound on the KL Divergence of Mixture Components

In this section, we derive the upper bound for the Kullback-Leibler divergence between a single-component distribution and the mixture distribution of which it is a part. This result justifies the necessity of controlling the reference mixture size to avoid score saturation.

Proposition B.1. *Let $m(x) = \sum_{j=1}^M w_j p_j(x)$ be a mixture of M probability density functions with weights $w_j \geq 0$ such that $\sum w_j = 1$. For any component $i \in \{1, \dots, M\}$, the KL divergence between p_i and the mixture m is bounded by:*

$$D_{\text{KL}}(p_i \parallel m) \leq -\log w_i. \quad (16)$$

Proof. Consider the ratio between the i -th component density and the total mixture density. By the definition of the mixture, we have:

$$m(x) = w_i p_i(x) + \sum_{j \neq i} w_j p_j(x).$$

Since density functions are non-negative ($p_j(x) \geq 0$) and the weights are non-negative, it follows that for all x in the support of p_i :

$$m(x) \geq w_i p_i(x) \implies \frac{p_i(x)}{m(x)} \leq \frac{1}{w_i}.$$

Taking the logarithm of both sides (noting that $\log$ is a monotonically increasing function), yields

$$\log \frac{p_i(x)}{m(x)} \leq \log \frac{1}{w_i} = -\log w_i.$$

The Kullback-Leibler divergence is defined as the expectation of this log-ratio over the distribution p_i :

$$D_{\text{KL}}(p_i \parallel m) = \int p_i(x) \log \frac{p_i(x)}{m(x)} dx \leq \int p_i(x) (-\log w_i) dx.$$

Since $-\log w_i$ is a constant with respect to x and the density $p_i(x)$ integrates to 1, we obtain:

$$D_{\text{KL}}(p_i \parallel m) \leq -\log w_i.$$

□

Application to Uniform Mixtures In the context of the policy population $\mathcal{D}_\Theta$, we assume a uniform mixture where $w_i = 1/M$ for all i . Substituting this into (16) yields:

$$D_{\text{KL}}(d_{\theta_i} \parallel d_\Psi) \leq -\log(1/M) = \log M.$$

This confirms that the maximum possible value of the divergence is constrained by the population size, leading to the saturation effect discussed in Section 4.1.

C. Numerically Stable Importance Weight Computation

To compute the importance weights without encountering numerical underflow, we transform the product of probabilities into a sum of log-probabilities. Let $\ell_{i,j}$ be the log-probability of trajectory τ_j under policy π_{θ_i} :

$$\ell_{i,j} = \log \Pr(\tau_j \mid \pi_{\theta_i}) = \sum_{t=0}^T \log \pi_{\theta_i}(a_t^j \mid s_t^j).$$

Similarly, $\ell'_{i,j}$ is defined for the target policies $\pi_{\theta'_i}$. The non-normalized weight $\bar{w}_j$ for the j -th trajectory is the ratio of two sums of exponentials. In log-space, this is expressed as:

$$\log \bar{w}_j = \log \left(\sum_i \exp(\ell'_{i,j}) \right) - \log \left(\sum_i \exp(\ell_{i,j}) \right).$$

Each term is computed using the *log-sum-exp* (LSE) function to prevent overflow/underflow:

$$\text{LSE}(x_1, \dots, x_n) = c + \log \sum_{i=1}^n \exp(x_i - c), \quad \text{where } c = \max_i \{x_i\}.$$

By extracting the maximum value c , we ensure that the largest value in the summation is $\exp(0) = 1$, effectively shifting the range of the calculation into a stable floating-point region.

The final step is self-normalization, which ensures that the weights over the dataset $\mathcal{D}_\tau$ sum to unity (i.e., $w_j = \bar{w}_j / \sum_k \bar{w}_k$), thereby constraining their variance. To maintain stability until the final step, we perform this normalization in the log-domain as well.

Let $\lambda_j = \log \bar{w}_j$ be the log-weight obtained from the previous step. The normalized weight w_j is computed as:

$$w_j = \exp(\lambda_j - \text{LSE}_k(\lambda_k)). \quad (17)$$

This approach ensures that we only exponentiate the values at the very end of the computation, preserving high numerical precision throughout the process.

D. Implementation details

In this section, we present additional information on the architectures, algorithms and their hyperparameters that we used to perform the experiments in Section 5.

D.1. Autoencoder Architecture and Training

Architecture Across all experiments, we used a fixed architecture with symmetric encoder and decoder networks:

- **Encoder:**
 - Input: Fully connected, n neurons
 - Layer 1: Fully connected, 25 neurons
 - Layer 2: Fully connected, 10 neurons
- **Bottleneck:**
 - Layer 3: Fully connected, k neurons
- **Decoder:**
 - Layer 4: Fully connected, 10 neurons
 - Layer 5: Fully connected, 25 neurons
 - Output: Fully connected, n neurons

Every layer uses Exponential Linear Unit (ELU) as activation function.

Autoencoder Training The training hyperparameters were tuned as follows:

- **Mini-batch Size:** We adopted small batch sizes ($P \in \{5, 10, 20\}$) to prevent the model from converging to a trivial mean-policy solution, forcing the manifold to capture the distinct behavioral modes of individual policies, especially the rare, high-reward trajectories that would otherwise be smoothed out. While this solution introduced significant gradient noise, we found it to be effective.
- **Number of trajectories:** We considered 20 trajectories for each policy when computing the mixture-matching loss function. We found this to be a good trade-off between the variance of the estimator and computational efforts.
- **Training Duration:** The number of iterations was calculated to provide a probabilistic coverage of the dataset. We ensured that the total iterations allow for each policy in $\mathcal{D}_\Theta$ to be sampled at least once with a probability greater than 0.99. We also found that random sampling of the policies, rather than simply iterating over the dataset, generally prevented catastrophic forgetting.
- **k-NN parameter:** Across all experiments, we fixed $k = 30$ for the importance sampling estimator. This value proved to be reliable across diverse policy populations and number of trajectories.

D.2. Policy Architecture

To model the policies, we use fully-connected multi-layer perceptrons (neural networks). The policy sizes and specific architectures considered are:

- **Small (~ 10 parameters):**
 - Input: Fully connected, s neurons, *ELU* activation
 - Layer 1: Fully connected, 4 neurons, *ELU* activation
 - Output: Fully connected, a neurons, tanh activation
- **Medium ($\sim 10^3$ parameters):**
 - Input: Fully connected, s neurons, *ELU* activation
 - Layer 1: Fully connected, 32 neurons, *ELU* activation
 - Layer 2: Fully connected, 32 neurons, *ELU* activation
 - Output: Fully connected, a neurons, tanh activation
- **Large ($\sim 4 \cdot 10^3$ parameters):**
 - Input: Fully connected, s neurons, *ELU* activation
 - Layer 1: Fully connected, 64 neurons, *ELU* activation
 - Layer 2: Fully connected, 64 neurons, *ELU* activation
 - Output: Fully connected, a neurons, tanh activation

D.3. Environments

As briefly explained in Section 5.4, we evaluate our methods on three control environments from the Gymnasium library:

Mountain Car Continuous (MC) This environment features a car stochastically placed in a sinusoidal valley. The objective is to build sufficient momentum to reach the goal state atop the right-hand hill.

- **State Space:** $\mathcal{S} \subseteq \mathbb{R}^2$, defined by the car’s position $p \in [-1.2, 0.6]$ and velocity $v \in [-0.07, 0.07]$.
- **Action Space:** $\mathcal{A} \subseteq \mathbb{R}^1$, representing the acceleration $a \in [-1, 1]$ applied to the vehicle.

- **Tasks and Rewards:** We define four distinct behavioral objectives:
 - **Standard:** A sparse reward of +100 upon reaching the right goal ($p \geq 0.45$), penalized by $-0.1a^2$ at each timestep.
 - **Left:** Identical to the standard task, but the goal is moved to the left peak ($p \leq -1.1$).
 - **Height:** A dense reward $R_t = h^2$ for $h \geq 0.2$, where $h = \sin(3p) \cdot 0.45 + 0.55$ as described by the environment’s geometry.
 - **Speed:** A dense reward $R_t = v^2$ provided at each timestep.
- The episode terminates upon reaching the standard goal or after 999 steps.

Reacher (RC) Reacher consists of a two-jointed robotic arm moving in a 2D plane toward a target. For our unsupervised framework, we remove all target-related information to focus on task-agnostic behavioral discovery.

- **State Space:** $\mathcal{S} \subseteq \mathbb{R}^6$, capturing the cosines and sines of the two joint angles ($\cos q_i, \sin q_i$) and their respective angular velocities ω_i . For normalization, the state is bounded by $\mathbf{s}_{min} = [-1, -1, -1, -1, -5, -5]$ and $\mathbf{s}_{max} = [1, 1, 1, 1, 5, 5]$.
 - **Action Space:** $\mathcal{A} \subseteq \mathbb{R}^2$, representing the torques applied to each hinge.
 - **Tasks and Rewards:** We define four binary behavioral tasks ($R_t = 1$ if the condition is met, 0 otherwise) based on the kinematics of the fingertip end-effector:
 - **Speed:** Linear tip velocity > 6 .
 - **Clockwise:** Tangential tip velocity < -11 .
 - **C-Clockwise:** Tangential tip velocity > 1 .
 - **Radial:** Radial tip velocity > 3 .
- Episodes terminate after a fixed horizon of 50 steps.

Hopper (HP) Hopper is a planar monopod robot with three joints (hip, knee, ankle). This environment is significantly more complex due to the requirement of maintaining balance while moving.

- **State Space:** $\mathcal{S} \subseteq \mathbb{R}^{11}$, capturing the robot’s vertical z -position, joint angles, and velocities. The state is normalized within the range $\mathbf{s}_{min} = [0.7, -0.2, -2.7, -2.7, -0.8, -5.0, \dots, -5.0]$ and $\mathbf{s}_{max} = [1.5, 0.2, 0.0, 0.0, 0.8, 5.0, \dots, 5.0]$.
- **Action Space:** $\mathcal{A} \subseteq \mathbb{R}^3$, representing the torques applied to the three hinges.
- **Tasks and Rewards:** As with Reacher, we define four behavioral tasks to characterize locomotion and agility. Three of these utilize a binary reward shape ($R_t \in \{0, 1\}$), while the fourth follows the Gymnasium formulation for the standard reward:
 - **Forward:** x -axis velocity > 1 .
 - **Backward:** x -axis velocity < -1 .
 - **Standstill:** x -axis velocity within $[-0.05, 0.05]$.
 - **Standard:** This task utilizes the environment’s default composite reward function:

$$R_t = R_{healthy} + w_f \frac{dx}{dt} - w_c \|a\|_2^2$$

where $R_{healthy}$ is a constant reward of 1 for each timestep the robot remains upright, $w_f \frac{dx}{dt}$ is the forward velocity of the torso, and $w_c \|a\|_2^2$ is a control penalty ($w_c = 10^{-3}$) that discourages excessively large actions.

Episodes terminate if the robot falls (e.g., its z -height or angle exceeds healthy bounds) or after 1,000 steps.

D.4. Hyperparameters

Baselines The baseline algorithms, PPO, DDPG, TD3, and SAC, were implemented using the StableBaselines library. Unless otherwise specified, we adhered to the library’s default parameterization.

In the **Mountain Car (MC)** environment, specific overrides were applied to improve exploration. For the off-policy methods **TD3** and **DDPG**, we increased the standard deviation of the exploration noise to 0.75 and 0.65, respectively. DDPG was also configured with a reduced replay buffer size of 50000. For **SAC**, we utilized generalized State Dependent Exploration (gSDE) and set the soft update coefficient (τ) to 0.01, the train frequency to 32, and the entropy coefficient to 0.1, performing 32 gradient steps per rollout with a buffer size of 50,000. The **PPO** implementation utilized gSDE with a learning rate of 10^{-4} , 32 steps per rollout, a batch size of 256, and 4 optimization epochs per rollout. Additional PPO parameters included a GAE λ of 0.9, a clip range of 0.1, an entropy coefficient of 0.1, a value function coefficient of 0.19, and a maximum gradient norm of 5.

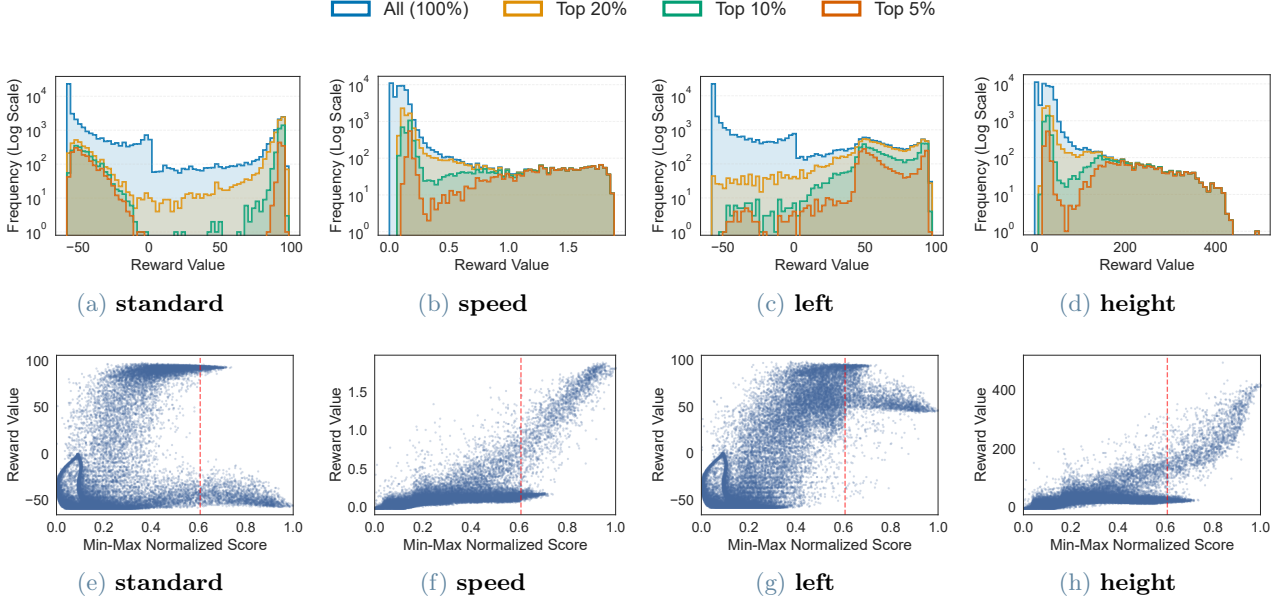


Figure 14: Mountain Car (MC), Small Policies

For the **Reacher** (RC) environment, the hyperparameters remained consistent with those of MC, with the sole exception of the **DDPG** exploration noise, which was reduced to 0.5. For the **Hopper** (HP) environment, all baselines were trained using the standard StableBaselines default parameters without modifications.

Latent PGPE During the Latent Optimization phase, we employed *Adam* as the underlying optimizer for PGPE. Across all environments, we utilized a population size of 8 and standard momentum parameters, with the exception of a reduced $\beta_1 = 0.1$ to avoid overshooting high reward areas and falling into a low reward plateau with virtually zero gradient. To ensure convergence, an exponential scheduler was used to decay the learning rate over time.

The specific tuning for the **Mountain Car** environment varied depending on the target reward. For the **standard** and **left** rewards, we set the center learning rate to 0.11 and the standard deviation learning rate to 0.1, with an initial standard deviation of 1.0. For the **speed** reward, we utilized a center learning rate of 0.02 and a standard deviation learning rate of 0.01. The **height** reward required a more conservative center learning rate of 0.01, combined with a standard deviation learning rate of 0.1 and a smaller initial standard deviation of 0.1.

For all tasks in the **Reacher** and **Hopper** environments, we utilized the same center learning rate of 0.15 and standard deviation learning rate of 0.12, as well as an initial standard deviation of 1.0.

E. Additional Experiments and Results

Here we present additional figures and plots that integrate the analyses presented in Section 5.

E.1. Reward Populations and Score Filtering

Here are displayed the reward distributions in the considered datasets and the effect that the score filtering has on them across every environment and, for the MC environment, different policy sizes. In all figures, the top row illustrates the log-frequency of rewards across different selection percentiles, and the bottom row shows the correlation between the uniqueness scores and task-specific rewards, with the vertical dashed line indicating the Top 5% selection threshold. We also provide an explanation for why, in some tasks, there is such a mismatch between the maximum reward achieved by the best policy in the dataset and maximum reward reached during Latent Optimization.

As illustrated by the Hopper reward distributions in Figure 18, while there exist specific policies capable of achieving high rewards in the **backward** and **forward** tasks (Figures 18a, 18b), they represent an extreme minority, constituting fewer than a dozen instances out of several thousand. Under the mixture-matching objective, the autoencoder’s primary goal is to represent the global distribution accurately; consequently, these high-performing outliers are ‘smoothed’ over by the latent bottleneck in favor of the more prominent, represen-

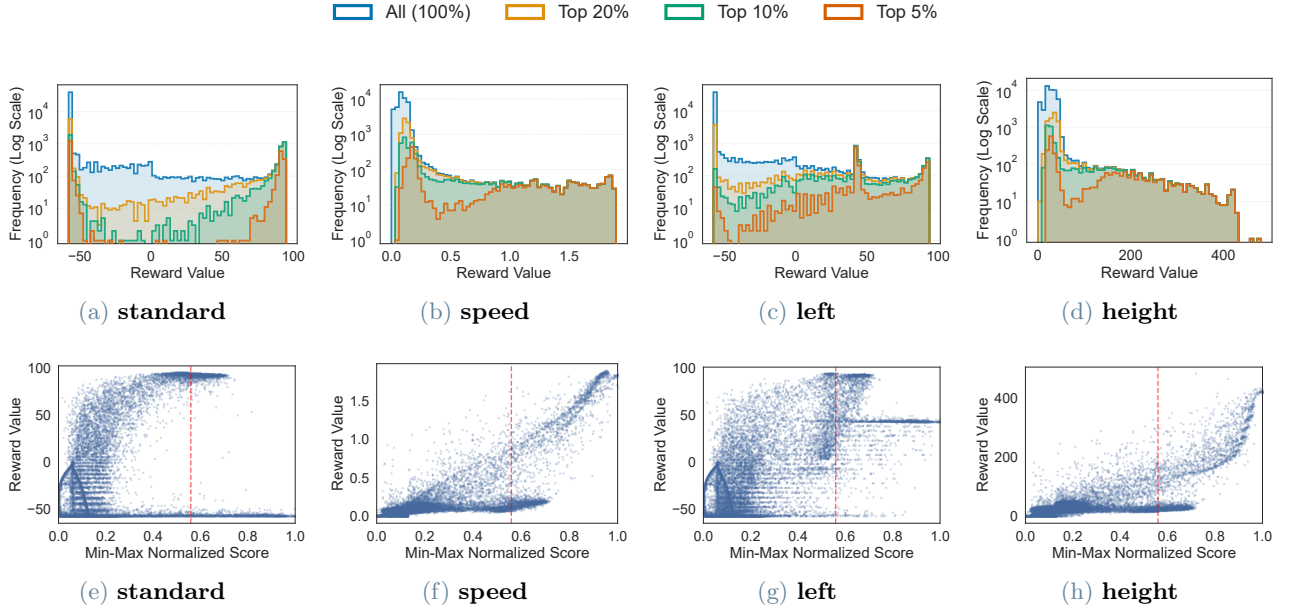


Figure 15: Mountain Car (MC), **Medium** Policies

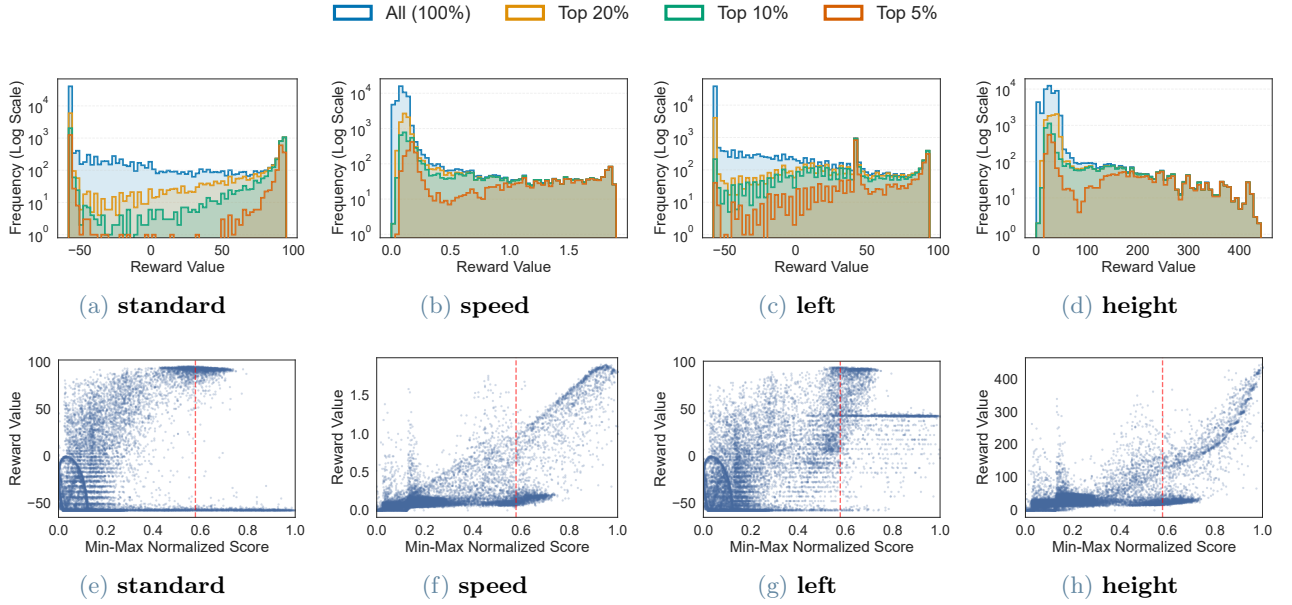


Figure 16: Mountain Car (MC), **Large** Policies

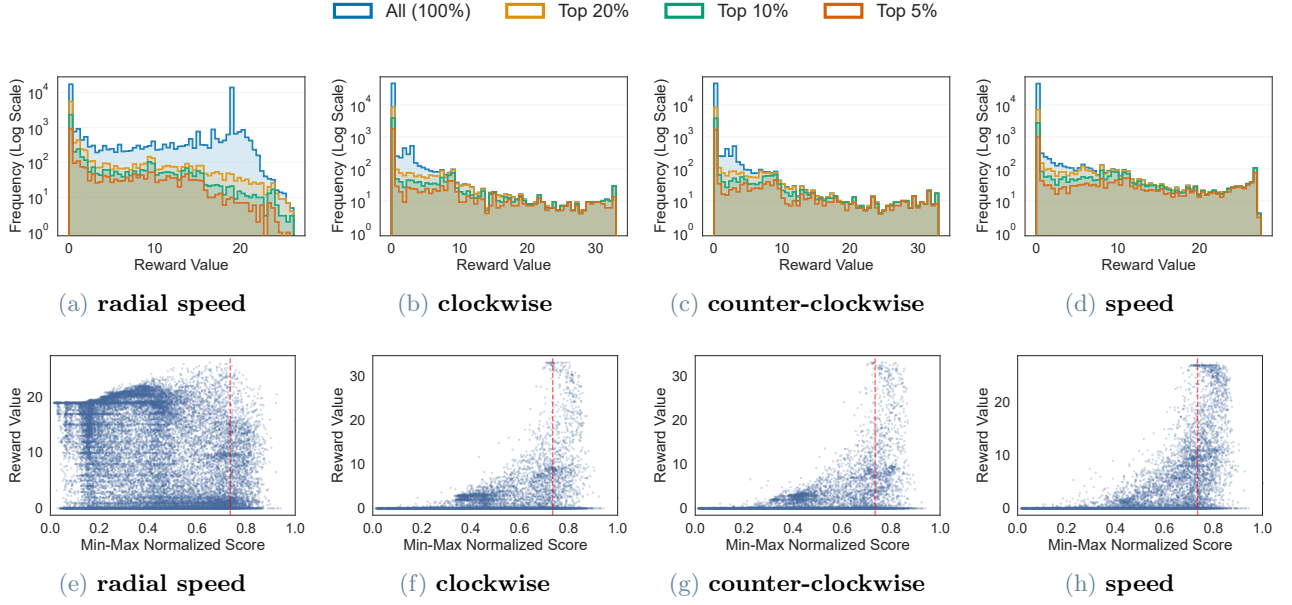


Figure 17: Reacher (RC), Medium Policies

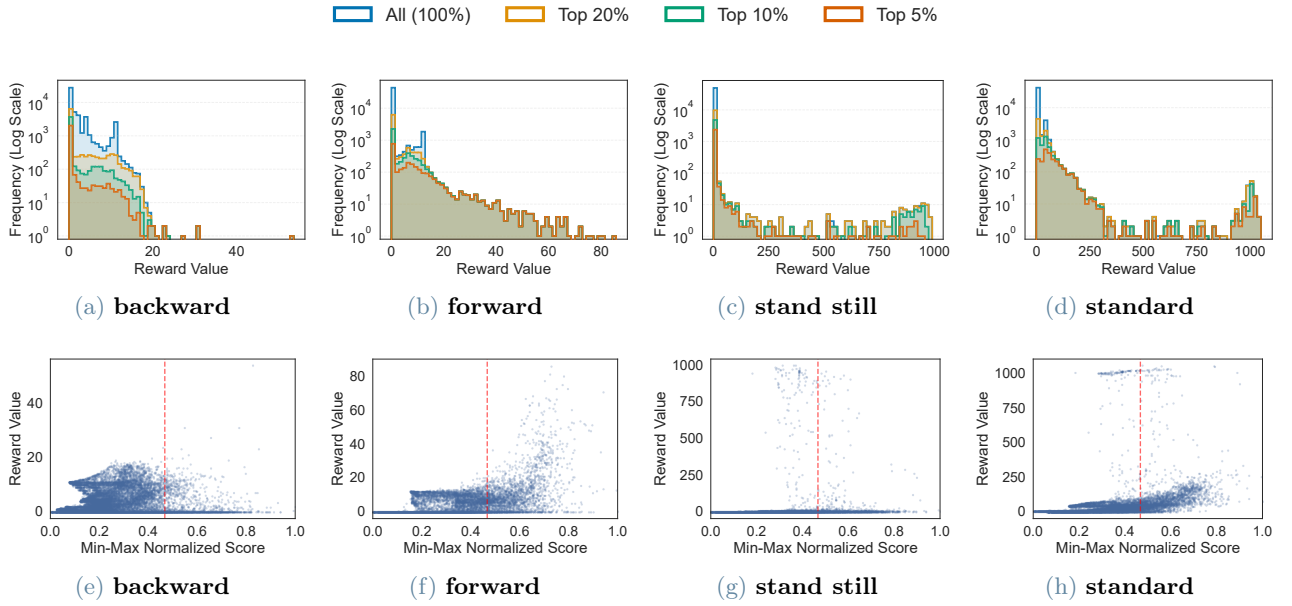


Figure 18: Hopper (HP), Medium Policies

tative behavioral modes. Additionally, it’s interesting to notice, as shown by the scatterplots (Figures 18e, 18f), how, despite these policies possessing the highest uniqueness scores in the dataset, their extremely low density prevents the latent space from dedicating sufficient capacity to their reconstruction. This confirms that the behavioral ceiling observed in our optimization results is a byproduct of this statistical smoothing rather than a failure of the search algorithm itself.

E.2. Latent Behavior Manifolds

In this section we display additional latent behavioral manifolds evaluated across different tasks. **Lighter** colors indicate higher rewards, while **darker** colors indicate lower rewards.

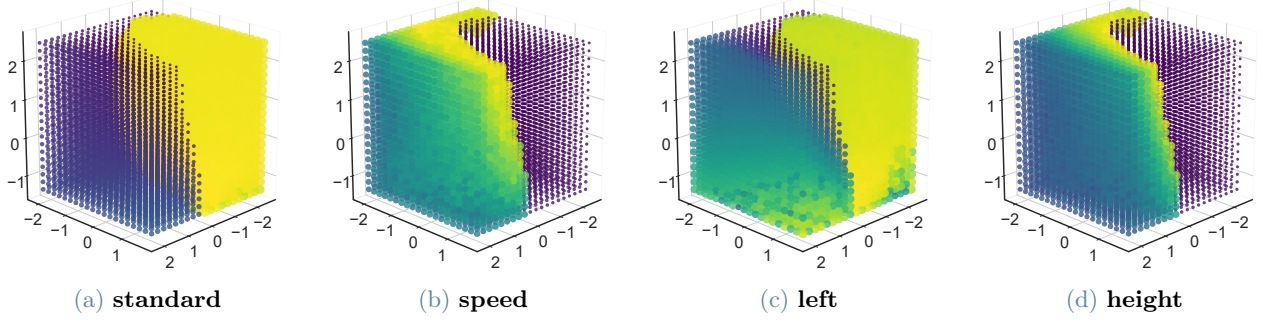


Figure 19: Visualization of the **Small** policies behavioral manifolds across different rewards for the **MC** environment.

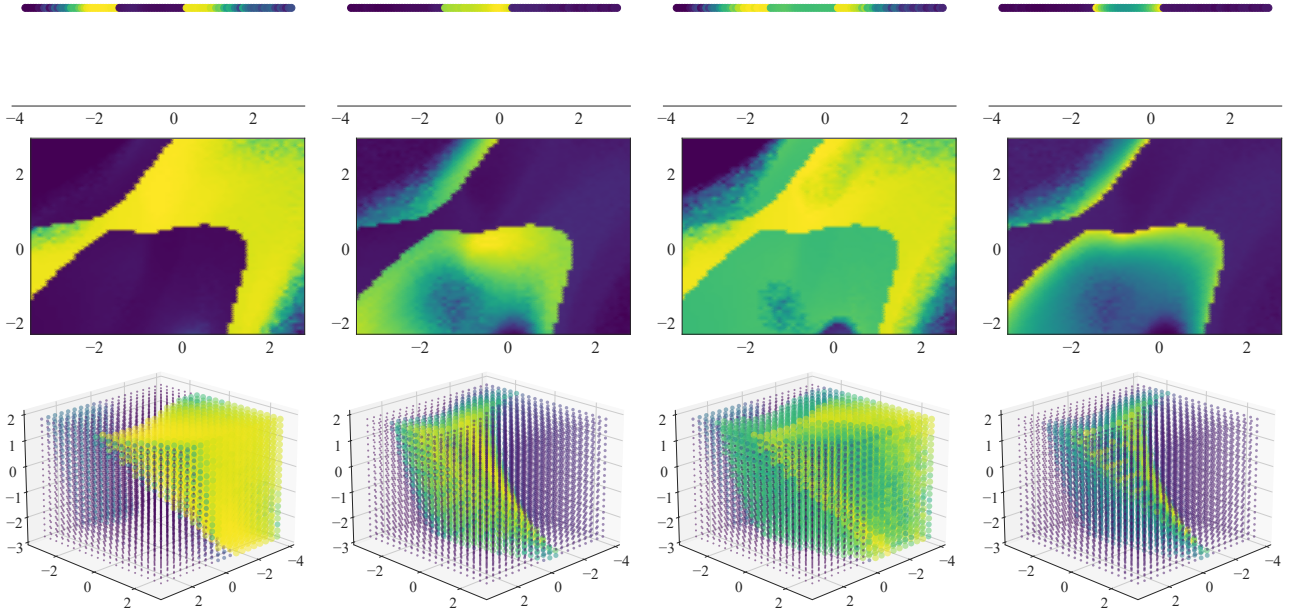


Figure 20: Visualization of the **Medium** policies behavioral manifolds across different rewards for the **MC** environment.

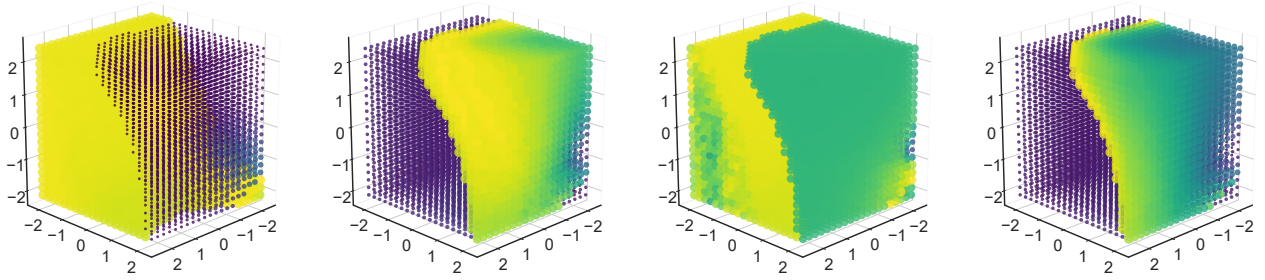


Figure 21: Visualization of the **Large** policies behavioral manifolds across different rewards for the **MC** environment.

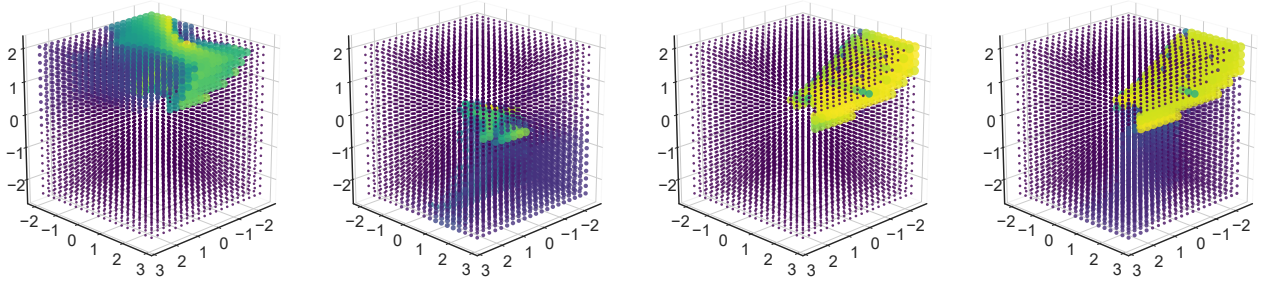


Figure 22: Visualization of the **Medium** policies behavioral manifolds across different rewards for the **HP** environment.

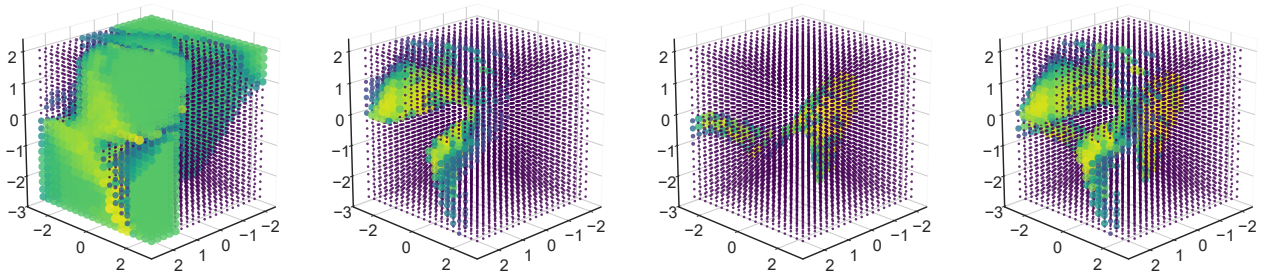


Figure 23: Visualization of the **Medium** policies behavioral manifolds across different rewards for the **RC** environment.

Abstract in lingua italiana

L'elevata dimensionalità e la ridondanza dello spazio dei parametri nel Deep Reinforcement Learning (DRL) rappresentano i principali ostacoli all'esplorazione e all'ottimizzazione delle policy. Questa tesi propone ed esplora un framework volto a separare la fase di scoperta dei comportamenti dalla loro ottimizzazione specifica, attraverso l'identificazione e la compressione della *varietà comportamentale* (behavioral manifold) di un agente. La metodologia si articola in tre fasi. Inizialmente, viene definita una metrica di unicità per filtrare una vasta popolazione di policy generate indipendentemente dall'ambiente. Successivamente, un'Autoencoder viene utilizzato per mappare lo spazio dei parametri in una rappresentazione latente più compatta. Al fine di garantire che la geometria latente rifletta la similarità dei comportamenti, il modello viene addestrato tramite una funzione obiettivo di matching dell'occupazione degli stati (state occupancy matching), utilizzando un approccio differenziabile basato su importance sampling. Infine, l'ottimizzazione del task specifico viene condotta nello spazio latente tramite l'algoritmo Policy Gradient with Parameter-based Exploration (PGPE).

Le valutazioni empiriche su benchmark di controllo classico e simulazioni MuJoCo dimostrano che il framework proposto garantisce una stabilità e un'efficienza campionaria (sample efficiency) superiore rispetto a diverse baseline DRL standard. I risultati indicano che i manifold appresi sono topologicamente regolari e invarianti rispetto alla dimensione delle reti neurali utilizzate. Viene tuttavia evidenziato un effetto di *latent smoothing*: sebbene il manifold offra uno spazio di ricerca utile, le prestazioni finali risultano vincolate dal repertorio dei comportamenti scoperto nella fase iniziale. Questa tesi pone le basi per un'ottimizzazione delle policy più efficiente, spostando il focus dai parametri ridondanti a rappresentazioni comportamentali più strutturate.

Parole chiave: apprendimento per rinforzo non supervisionato, compressione di politica, manifold comportamentale, matching delle occupazioni