# Unsupervised Behavioral Compression: Learning Low-Dimensional Policy Manifolds through State-Occupancy Matching

**Andrea Fraschini**[♣,†], **Davide Tenedini**[♣,†], **Riccardo Zamboni**[♣], **Mirco Mutti**[♠], **Marcello Restelli**[♣]

andrea.fraschini@mail.polimi.it,
{davide.tenedini, riccardo.zamboni, marcello.restelli}@polimi.it,
mirco.m@technion.ac.il

♣ **Politecnico di Milano**
♠ **Technion**

† Equal contribution

## Abstract

Deep Reinforcement Learning (DRL) is widely recognized as sample-inefficient, a limitation attributable in part to the high dimensionality and substantial functional redundancy inherent to the policy parameter space. A recent framework, which we refer to as Action-based Policy Compression (APC), mitigates this issue by compressing the parameter space $\Theta$ into a low-dimensional latent manifold $\mathcal{Z}$ using a learned generative mapping $g : \mathcal{Z} \to \Theta$. However, its performance is severely constrained by relying on immediate action-matching as a reconstruction loss, a myopic proxy for behavioral similarity that suffers from compounding errors across sequential decisions. To overcome this bottleneck, we introduce Occupancy-based Policy Compression (OPC), which enhances APC by shifting behavior representation from immediate action-matching to long-horizon state-space coverage. Specifically, we propose two principal improvements: (1) we curate the dataset generation with an information-theoretic uniqueness metric that delivers a diverse population of policies; and (2) we propose a fully differentiable compression objective that directly minimizes the divergence between the true and reconstructed mixture occupancy distributions. These modifications force the generative model to organize the latent space around true functional similarity, promoting a latent representation that generalizes over a broad spectrum of behaviors while retaining most of the original parameter space's expressivity. Finally, we empirically[1] validate the advantages of our contributions across multiple continuous control benchmarks.

## 1 Introduction

The success of Deep Reinforcement Learning (DRL) in complex continuous control relies heavily on the highly expressive nature of deep neural networks (Lillicrap et al., 2016; Schulman et al., 2017). However, optimizing these high-dimensional policies *tabula rasa* is notoriously sample-inefficient (Duan et al., 2016; Agarwal et al., 2022). This inefficiency stems in part from a fundamental functional redundancy: Due to the complex symmetries of neural architectures, vast and disparate regions of the parameter space often collapse into identical agent behaviors. Searching this massive, unstructured space blindly ignores the underlying functional geometry of the environment. This

[1] The source code is available at this link.

limitation becomes a severe bottleneck in multi-task or Unsupervised Reinforcement Learning (URL) settings (Laskin et al., 2021), where an agent must train a model that rapidly adapts to a variety of previously unknown downstream tasks rather than learning from scratch.

To address this redundancy in the policy parameter space, prior work has proposed various approaches to compress the policy space onto lower-dimensional latent manifolds (e.g., Eysenbach et al., 2018; Mutti et al., 2022; Faccio et al., 2023; Tenedini et al., 2026; Li et al., 2026). In this stream, `Action-based Policy Compression` (APC, Tenedini et al., 2026) explicitly shifts the learning paradigm from parameter optimization to behavior optimization. APC leverages a generative model to compress a highly redundant policy parameter space into a low-dimensional, behaviorally structured latent manifold. By curating a diverse dataset of random policies and training an autoencoder to reconstruct them, agents can subsequently bypass the raw parameter space entirely. Instead, they perform task-specific fine-tuning by directly navigating the compact latent space. By following this recipe, APC compresses policy architectures with several thousand parameters into tiny two- or three-dimensional latent spaces, while retaining most of the functional expressivity to address a variety of tasks in classical continuous control domains, such as Mountain Car and Reacher.

While conceptually and empirically promising, the practical adaptability of APC beyond these domains is severely constrained by the technical solutions adopted to train the latent space. In particular, to organize this latent space functionally rather than geometrically, APC relies on an action-matching objective, training the autoencoder to minimize the discrepancy between the immediate action outputs of the original and reconstructed policies. However, measuring behavioral similarity through immediate action divergence is a mathematically loose and myopic proxy for long-horizon control (Metelli et al., 2018). Because errors compound across sequential decisions, policies that appear functionally identical at the single-action level can lead to drastically different trajectories and global state visitations. Furthermore, Tenedini et al. (2026) curate the initial pre-training dataset of policies using novelty search in the action space, which frequently discards rare, high-performing behaviors, severely impoverishing the topological diversity of the resulting manifold.

To overcome these limitations, we introduce `Occupancy-based Policy Compression` (OPC), representing a paradigm shift in latent manifold learning for DRL. We abandon immediate action-matching entirely in favor of global state-space coverage. Specifically, we introduce two principal algorithmic enhancements. First, we curate the pre-training dataset using an information-theoretic uniqueness metric that actively selects policies based on their contribution to the global state-visitation entropy, retaining a much broader spectrum of rare behaviors. Second, we propose a fully differentiable compression objective that explicitly matches the true mixture occupancy distribution of the policy population. By evaluating policies based on the exact states they visit over the full horizon, OPC completely bypasses the compounding errors of action-matching, forcing the generative model to organize the latent space around true, long-horizon functional similarity.

Then, to support the design choices behind OPC, we address the following:

**Research Questions:**

(**Q1**) Does curating policies based on their contribution to state-space coverage preserve a wider spectrum of rare, high-performing behaviors compared to action-space novelty search?

(**Q2**) Does optimizing a compression objective based on state-space visitation patterns yield a more behaviorally meaningful latent space than action-based behavioral reconstruction?

(**Q3**) Does navigating an occupancy-matched latent space improve sample efficiency and task performance compared to standard baselines and action-matched manifolds?

**Content Outline and Contributions.** In Section 2, we establish the necessary notation. In Section 3, we detail the existing APC framework and expose the limitations of action-matching. In Section 4, we introduce our OPC framework. Finally, in Section 5, we address our Research Questions with extensive empirical evidence, demonstrating that OPC retains richer behavioral modes, constructs superior latent topologies, and enables fast few-shot adaptation to various downstream tasks.

## 2 Preliminaries

**Notation.** For a set $\mathcal{A}$ of finite size $|\mathcal{A}|$, we denote $\Delta(\mathcal{A})$ its probability simplex. For two distributions $p, q$, the Kullback-Leibler (KL) divergence is $D_{\mathrm{KL}}(p \parallel q)$ and the differential entropy of $p$ is $H(p)$.

**Interaction Protocol.** We model the interactions between the agent and the environment as a (finite-horizon) **Controlled Markov Process** (CMP). A CMP is a tuple $\mathcal{M} := (\mathcal{S}, \mathcal{A}, \mathbb{P}, \mu, T)$, where $\mathcal{S}$ is the state space and $\mathcal{A}$ is the action space. An episode begins with the initial state $s_0 \sim \mu \in \Delta(\mathcal{S})$, where $\mu$ is the initial state distribution. At each timestep $t = 0, \ldots, T$, where $T < \infty$ is the episode horizon, the agent observes the current state $s_t$, takes action $a_t \in \mathcal{A}$, such that the model transitions to the next state $s_{t+1} \sim \mathbb{P}(\cdot \mid s_t, a_t)$ according to the transition model $\mathbb{P} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$. The agent acts according to a *stochastic policy* $\pi : \mathcal{S} \to \Delta(\mathcal{A})$ such that $\pi(a \mid s)$ denotes the conditional probability of taking action $a$ in state $s$. This interaction induces a trajectory $\tau := (s_0, a_0, s_1, a_1, \ldots, s_T)$.

**Occupancy Measure.** The deployment of policy $\pi$ in $\mathcal{M}$ induces a marginal distribution over the state space, known as the (state) *occupancy measure* $d_\pi \in \Delta(\mathcal{S})$, which represents the average visitation probability of state $s$ over the episode length: $d_\pi(s) = \frac{1}{T+1} \sum_{t=0}^{T} \Pr(s_t = s \mid \pi)$. If we consider the deployment of a population of $M$ distinct policies $\Psi = \{\pi_1, \ldots, \pi_M\}$, we define the **mixture occupancy measure** $d_\Psi \in \Delta(\mathcal{S})$ as the (uniform) mixture of individual occupancies: $d_\Psi(s) = \frac{1}{M} \sum_{i=1}^{M} d_{\pi_i}(s), \quad \forall s \in \mathcal{S}$.

We consider parametric policies $\pi_{\boldsymbol{\theta}}$ represented by neural networks parametrized by $P$-dimensional weight vectors $\boldsymbol{\theta} \in \Theta \subseteq \mathbb{R}^P$, where $\Theta$ represents the policy parameter space. We denote the collection of representable policies as the *Policy Space* $\Pi_\Theta$. As shorthand, we use the parameter vector $\boldsymbol{\theta}$ in place of its induced policy $\pi_{\boldsymbol{\theta}}$ and the parameter space $\Theta$ in place of the policy space $\Pi_\Theta$.

**Non-Parametric Estimation.** In continuous spaces, a direct computation of the $D_{\mathrm{KL}}(f \parallel f')$ between distributions $f, f'$ is often intractable. Other approaches, such as *particle-based estimators*, are used to approximate these quantities using the finite set of states of collected trajectories. Specifically, Ajgl & Šimandl (2011) propose an **importance sampling $k$-NN estimator**:

$$\widehat{D}_{KL}(f \parallel f') = \frac{1}{N} \sum_{i=1}^{N} \ln \frac{k/N}{\sum_{j \in \mathcal{N}_i^k} w_j}, \quad w_j = \frac{f'(x_j)/f(x_j)}{\sum_{n=1}^{N} f'(x_n)/f(x_n)}, \tag{1}$$

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

## 3 Background: Action-based Policy Compression

In Reinforcement Learning (RL), an agent interacts with a CMP equipped with a given reward signal $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, augmenting the CMP into a Markov Decision Process (MDP, Puterman, 2014) $\mathcal{M}_R := \mathcal{M} \cup R$. The agent's goal is typically to find a high-dimensional parameter vector $\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta} \in \Theta} J^R(\boldsymbol{\theta})$ that maximizes the expected cumulative sum of rewards $J^R(\boldsymbol{\theta})$ obtained by playing the policy $\pi_{\boldsymbol{\theta}}$. This process is notoriously sample-inefficient, partially due to a functional *redundancy*: vast, distinct regions of the parameter space $\Theta$ collapse into identical agent behaviors. As first noted by Tenedini et al. (2026), this inefficiency is severely exacerbated in **Unsupervised Reinforcement Learning** (URL, Laskin et al., 2021): the agent learns a task-agnostic representation $\mathbb{M}$ via *unsupervised pre-training* on a reward-free CMP, then leverages it during *supervised fine-tuning* to rapidly optimize the newly revealed reward $R$.

**The APC Framework.** This redundancy in URL was first addressed directly via **Action-based Policy Compression** (APC, Tenedini et al., 2026): the high-dimensional policy parameter space $\Theta \subseteq \mathbb{R}^P$ is compressed into a compact, behaviorally structured latent space $\mathcal{Z} \subseteq \mathbb{R}^k$, with $k \ll P$. By shifting the downstream search space from parameters to behaviors, APC enables efficient task adaptation. The framework operates in three sequential stages:

*1. Policy Dataset Curation:* To model the behavior manifold, APC generates a large pool of policies via random weight initialization. This naïve approach is computationally light, but prone to generating redundant, low-energy policies. To filter this pool into a diverse training dataset, APC employs a $k_n$-nearest-neighbor novelty search in the *action space*, selecting policies that produce diverse immediate action distributions across a set of collected states. Formally, they threshold the dataset to keep only the top percentile of policies according to the following score:

$$\rho^{\text{APC}}(\pi_{\boldsymbol{\theta}}) = \frac{1}{k_n} \sum_{i \in \mathcal{N}^{k_n}} D_{\text{KL}}(\pi_{\boldsymbol{\theta}} \parallel \pi_{\boldsymbol{\theta}_i}). \tag{2}$$

*2. Latent Behavior Compression:* APC defines the objective of Policy Compression as finding a generative mapping $g^* : \mathcal{Z} \to \Theta$, such that:

$$\forall \boldsymbol{\theta} \in \Theta, \quad \exists \boldsymbol{z} \in \mathcal{Z} : \quad g^\star = \arg\min_g D_{\text{KL}}\left(d^{sa}_{\pi_{\boldsymbol{\theta}}} \parallel d^{sa}_{\pi_{\boldsymbol{z}}}\right), \tag{3}$$

where $d^{sa}_\pi = \pi(a \mid s) \cdot d^s_\pi$ is the (state–action) occupancy measure. To achieve such an objective, APC employs an Autoencoder (AE, Hinton & Salakhutdinov, 2006), consisting of an encoder $f_{\boldsymbol{\xi}} : \Theta \to \mathcal{Z}$ and a decoder $g_{\boldsymbol{\zeta}} : \mathcal{Z} \to \Theta$, parametrized by $\boldsymbol{\xi}$ and $\boldsymbol{\zeta}$, respectively, trained to minimize the divergence between immediate actions of the original and reconstructed policy, defined as $\mathcal{L}_B(\boldsymbol{\xi}, \boldsymbol{\zeta}) = \mathbb{E}_{\boldsymbol{\theta} \sim \mathcal{D}_\Theta}\left[D_{\text{KL}}\left(\pi_{\boldsymbol{\theta}} \parallel \pi_{(g_{\boldsymbol{\zeta}} \circ f_{\boldsymbol{\xi}})(\boldsymbol{\theta})}\right)\right]$. This serves as a computationally efficient proxy to Eq. 3.

*3. Latent Behavior Optimization:* Once the decoder $g_{\boldsymbol{\zeta}}$ is trained and frozen, it serves as the task-agnostic representation $\mathbb{M}$ for fine-tuning. For a downstream task $R$, the agent searches for an optimal code $\boldsymbol{z}^* = \arg\max_{\boldsymbol{z} \in \mathcal{Z}} J^R(\boldsymbol{\theta} = g_{\boldsymbol{\zeta}}(\boldsymbol{z}))$. Because $\mathcal{Z}$ is low-dimensional, APC employs **Policy Gradient with Parameter-based Exploration** (PGPE, Sehnke et al., 2008). PGPE treats the decoder as a black box, avoiding costly Jacobian computations by sampling latent codes $\boldsymbol{z}$ from a hyper-policy $\nu_{\boldsymbol{\phi}}$ (e.g., a Gaussian distribution parameterized by $\boldsymbol{\phi} = (\boldsymbol{\mu}, \boldsymbol{\sigma})$). The hyperparameters are updated via a Monte Carlo gradient estimator over $N$ trajectories:

$$\hat{\nabla}_{\boldsymbol{\phi}} J^R(\boldsymbol{\phi}) = \frac{1}{N} \sum_{i=1}^{N} \nabla_{\boldsymbol{\phi}} \log \nu_{\boldsymbol{\phi}}(\boldsymbol{z}_i) R(\tau_i). \tag{4}$$

**Limitations: The Action-matching Bottleneck.** While APC successfully reduces the dimensionality of the policy space, its reliance on immediate action-matching poses a fundamental limitation for long-horizon continuous control. Theoretically, APC justifies this proxy by relying on the fact that bounding immediate action divergence also bounds the true discrepancy between state occupancies. Metelli et al. (Prop. E.1, 2018) provide an upper bound to this quantity for $H$-horizon settings, namely $H \sup_{s \in \mathcal{S}} D_{\text{KL}}(\pi_{\boldsymbol{\theta}}(\cdot \mid s) \parallel \pi_{\boldsymbol{\theta}'}(\cdot \mid s))$. In practice, due to compounding errors across sequential decisions, policies with near-identical action distributions can yield drastically different state-occupancy measures. This myopic behavioral representation in both the dataset curation and behavior compression phases hinders APC's ability to retain rare, but highly structured behaviors, severely impoverishing the topological diversity of the resulting latent manifold. Overcoming these limitations requires a paradigm shift: designing and optimizing behaviors based on long-horizon state-space coverage, rather than immediate actions.

## 4 Methodology: Occupancy-based Policy Compression

To overcome the inherent limitation of action-matching proxies, we introduce **Occupancy-based Policy Compression** (OPC). While retaining APC's three-stage pipeline, we introduce fundamental algorithm shifts to the dataset curation, manifold learning, and fine-tuning phases:

*1. Policy Dataset Curation:* Like APC, we initially generate a large candidate pool of policies $\Psi$ via random weight initialization. However, rather than using a $k$-NN search in the action space, we propose an information-theoretic selection criterion. We aim to identify policies that maximize the

coverage of the feasible state space. To do so, we score each candidate policy based on its contribution to the differential entropy of the mixture occupancy distribution $H(d_\Psi) = \frac{1}{M}\sum_{i=1}^{M} \rho^{\text{OPC}}(\boldsymbol{\theta}_i)$, where each component is defined as:

$$\rho^{\text{OPC}}(\boldsymbol{\theta}_i) = \left[H(d_{\boldsymbol{\theta}_i}) + D_{\text{KL}}(d_{\boldsymbol{\theta}_i} \parallel d_\Psi)\right], \quad \forall \boldsymbol{\theta}_i \in \Psi. \tag{5}$$

Intuitively, this metric selects policies that exhibit both high individual exploration (high entropy $H(d_{\boldsymbol{\theta}_i})$) and are statistically diverse from the population average (high divergence from the mixture $d_\Psi$). To make this computationally tractable for large populations, we approximate each occupancy distribution $d_{\boldsymbol{\theta}_i}$ with a Gaussian Mixture Model (GMM) fitted on trajectories collected in a trajectory dataset $\mathcal{D}_\tau$. A complete derivation and tractable estimators of $\rho$ can be found in Appendix B. The final training dataset $\mathcal{D}_\Theta$ is curated by selecting the top-scoring policies from $\Psi$ according to $\rho$.

*2. Latent Behavior Compression:* In the compression phase, OPC redefines the fundamental theoretical objective of Policy Compression. The original APC framework aimed for strict, 1-to-1 state-action occupancy matching (Eq. 3). However, forcing a generative model to perfectly reproduce individual policies in isolation is overly restrictive and discourages the discovery of a smooth, continuous behavioral topology. Instead, we argue that the true goal of manifold learning should be to preserve the **global state-space coverage** of the dataset as a whole. Therefore, OPC shifts the objective from individual matching to **mixture-occupancy matching**. We seek a generative mapping $g^\star$ and a corresponding population of latent codes $Z \in \mathcal{Z}^{|\Psi|}$ that minimizes the divergence between the mixture occupancy of the original population $\Psi$ and of the reconstructed population $g(Z) = \{g(\boldsymbol{z}) \mid \boldsymbol{z} \in Z\}$:

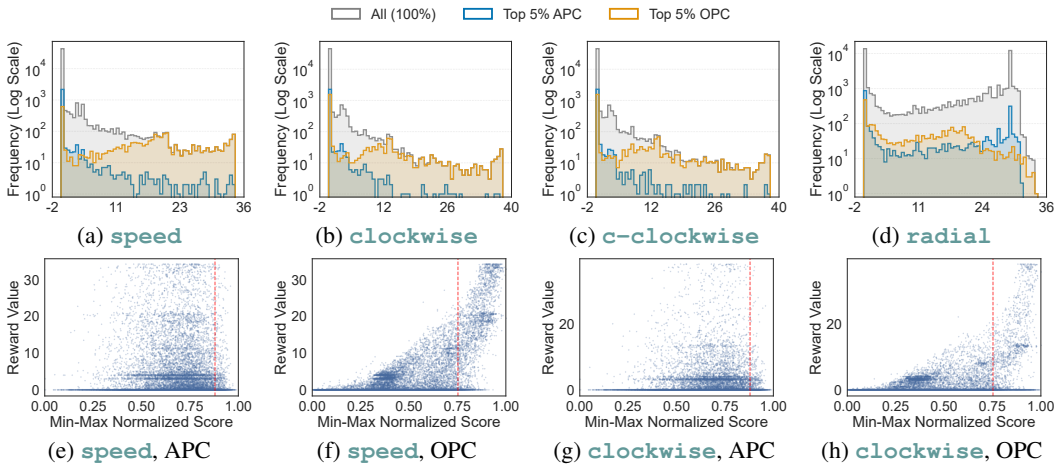
$$\exists \;\; Z \in \mathcal{Z}^{|\Psi|} : \quad g^\star = \arg\min_g D_{\text{KL}}(d_\Psi \parallel d_{g(Z)}). \tag{6}$$

By relaxing the strict 1-to-1 reconstruction constraint, this objective allows the generative model to organize the latent space purely around global, task-agnostic behaviors. Furthermore, unlike APC, which falls back on an action-matching proxy to approximate its objective, we can optimize Eq. 6 directly. The autoencoder minimizes the empirical loss $\mathcal{L}_B(\boldsymbol{\xi}, \boldsymbol{\zeta}) = D_{\text{KL}}(d_{\mathcal{D}_\Theta} \parallel d_{\mathcal{D}_{\widehat{\Theta}}})$, where $\mathcal{D}_{\widehat{\Theta}} = (g_{\boldsymbol{\zeta}} \circ f_{\boldsymbol{\xi}})(\mathcal{D}_\Theta)$. We achieve this end-to-end by estimating the divergence using the differentiable non-parametric importance sampling $k$-NN estimator defined in Eq. 1, evaluated over the trajectory dataset $\mathcal{D}_\tau$. The core of this estimator relies on the unnormalized importance weight $\overline{w}_j$ for a given trajectory $\tau_j = (s_0^j, a_0^j, \ldots, s_T^j, a_T^j)$, representing the density ratio between the reconstructed and original mixtures, computed as in Mutti et al. (2021):

$$\overline{w}_j = \frac{\sum_{\hat{\boldsymbol{\theta}} \in \mathcal{D}_{\widehat{\Theta}}} p(\tau_j \mid \hat{\boldsymbol{\theta}})}{\sum_{\boldsymbol{\theta} \in \mathcal{D}_\Theta} p(\tau_j \mid \boldsymbol{\theta})} = \frac{\sum_{\hat{\boldsymbol{\theta}} \in \mathcal{D}_{\widehat{\Theta}}} \prod_{t=0}^{T} \pi_{\hat{\boldsymbol{\theta}}}(a_t^j \mid s_t^j)}{\sum_{\boldsymbol{\theta} \in \mathcal{D}_\Theta} \prod_{t=0}^{T} \pi_{\boldsymbol{\theta}}(a_t^j \mid s_t^j)},$$

where $p(\tau \mid \boldsymbol{\theta}) = \mu(s_0) \prod_{t=0}^{T} \mathbb{P}(s_{t+1} \mid s_t, a_t)\, \pi_{\boldsymbol{\theta}}(a_t \mid s_t)$ is the probability density of a trajectory and $\hat{\boldsymbol{\theta}}$ is the vector of reconstructed parameters. Crucially, because the environment dynamics $\mathbb{P}$ and initial state distribution $\mu$ are identical for both the original and reconstructed policies, they cancel out entirely. This yields a purely policy-explicit formulation, rendering the loss function fully differentiable with respect to the Autoencoder parameters $\boldsymbol{\xi}$ and $\boldsymbol{\zeta}$. The final weights are obtained via self-normalization, $w_j = \overline{w}_j / \sum_{i=1}^{N} \overline{w}_i$, and by assigning to each particle $s_i^j$ the weight $w_j$ of the trajectory $\tau_j$ that produced it. A numerically stable formulation of the importance weights can be found in Appendix C. To manage the computational complexity of the $k$-NN estimator over large populations, we adopt a stochastic approach, computing the loss and updating the parameters over mini-batches of policies $\psi \subset \mathcal{D}_\Theta$ and their corresponding trajectories. The detailed algorithm for training the autoencoder is provided in Appendix D.

*3. Latent Behavior Optimization:* Similar to APC, we employ PGPE to optimize the latent code $\boldsymbol{z}$ directly, avoiding analytical computation of the Jacobian of the decoder. To fully exploit the low dimensionality and topologically dense structure of the OPC latent space, we introduce a **Latent Warm Start** phase prior to running PGPE. We first sample a small set of latent codes. We then




(a) `speed` (b) `clockwise` (c) `c-clockwise` (d) `radial`

(e) `speed`, APC (f) `speed`, OPC (g) `clockwise`, APC (h) `clockwise`, OPC

Figure 1: Comparison of APC and OPC thresholding in the RC environment: reward distribution in the curated datasets (top row, frequency is on a log scale) and score distribution comparison (bottom row, scores are min-max normalized). The vertical dashed line marks the 5% cut-off.

evaluate the reconstructed policies and use the best-performing latent code as a starting point for standard PGPE (Eq. 4). This coarse-to-fine strategy prevents PGPE from converging to poor local optima and accelerates fine-tuning by starting the gradient updates in a high-reward basin of attraction.

In the following section, we empirically validate how these structural enhancements translate to behaviorally richer learned latent manifolds.

## 5 Experiments

In this section, we empirically evaluate the proposed framework. We first outline the experimental setup and domains, then systematically address the three research questions introduced in Section 1.

**Experimental Settings.** We select continuous control domains that are challenging yet interpretable to clearly illustrate the features of the learned behavioral manifold. The first is `Mountain Car Continuous` (MC, Moore, 1990). To evaluate the quality of the latent space, we define four downstream tasks: `standard` and `left` place the goal state on the right and left hill, respectively; `speed` and `height` incentivize the car to maintain high velocity and elevation ($z$-coordinate), respectively, without terminating the episode. We also consider two MuJoCo environments (Todorov et al., 2012). In `Reacher` (RC), a two-jointed robotic arm moving in a 2D plane, we remove target observations and define four tasks based on fingertip kinematics: `speed` promotes high linear velocity; `clockwise` and `c-clockwise` promote sustained rotation; and `radial` promotes fast arm extension and retraction. Finally, in `Hopper` (HP), we evaluate locomotion via four downstream tasks: `forward`, `backward`, and `standstill` reward the agent for positive, negative, and zero velocity along the $x$-axis, respectively, while `standard` corresponds to the original Gymnasium composite reward. For all experiments, we generate a dataset of $50,000$ policies and retain the top 5% most informative ones. Policies are fully-connected neural networks with $\approx 10^3$ parameters. Full experimental details are provided in Appendix E.

### 5.1 Policy Dataset Curation

**Q1:** On the role of curating policies for improved datasets.

To answer **Q1**, we analyze the score distributions assigned to the policies and the reward distributions of the resulting datasets obtained during the curation phase. We compare the reward distribution of the original random dataset (100%) against a 5% thresholding according to both scores $\rho^{\text{APC}}$ (Eq. 2) and $\rho^{\text{OPC}}$ (Eq. 5). We also map the min–max normalized scores against the rewards across all

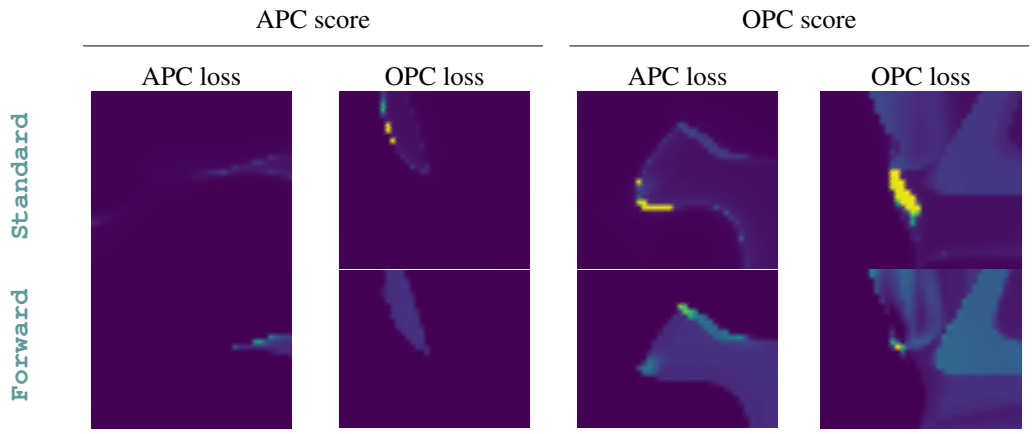



Figure 2: Ablation study comparing Latent Space topologies in the HP environment. Rows represent reward functions, while columns represent different score/loss combinations. **Lighter** and **darker** colors indicate higher and lower returns of the decoded policy.

tasks and environments. The core results for the RC environment are presented in Fig. 1, while the extended experimental results for all other domains are detailed in Appendix F.1.

As illustrated in Fig. 1a,1b,1c,1d, the random initialization process naturally yields a heavy-tailed distribution where high-reward behaviors are exceedingly rare; this phenomenon is exacerbated in complex control tasks such as Hopper. The APC novelty search, relying strictly on immediate action differences, frequently discards these rare behavioral modes. In contrast, the OPC curation successfully isolates and preserves these critical outlier policies, maintaining a significantly heavier tail in the high-reward regions across the evaluated tasks. This effect is explained by the score distributions: while scores $\rho^{\text{APC}}$ (Fig. 1e,1g) are generally uncorrelated with the actual task rewards or global behavioral uniqueness, scores $\rho^{\text{OPC}}$ (Fig. 1f,1h) are highly structured, ensuring that the most functionally unique policies consistently fall within the top percentile of the scoring distribution.

### 5.2 Latent Behavior Compression

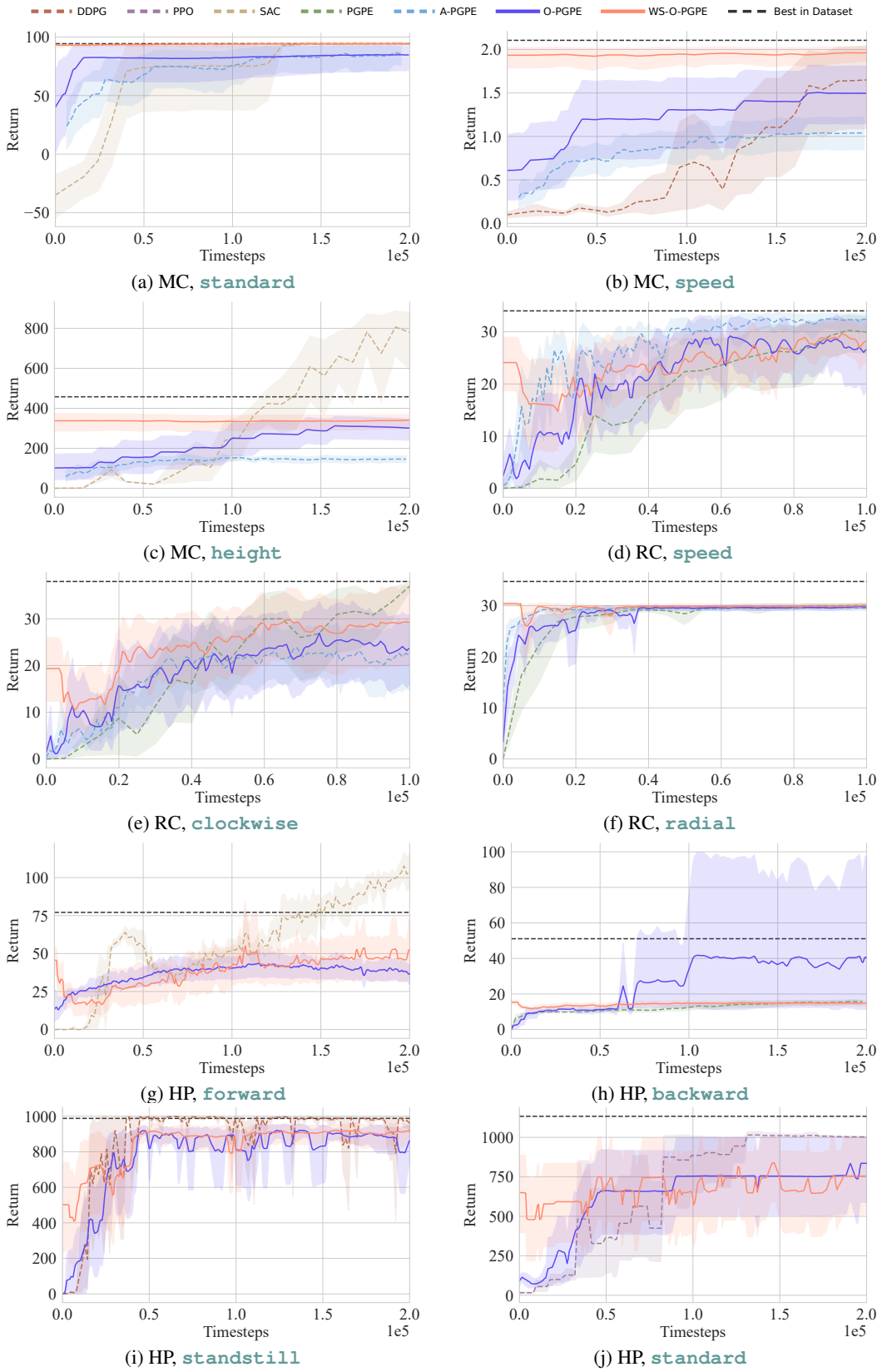
**Q2:** On the role of OPC in the quality of latent spaces.

To answer **Q2**, we perform a qualitative visual inspection of the latent spaces generated by both APC and OPC pipelines. We conduct an ablation study to assess the effects of the occupancy-based components (uniqueness scores and the compression objective). We focus on the HP environment, as it offers a more complex locomotion task. We fix the AE bottleneck size to 2 to provide clear visualizations. To achieve better performances in Section 5.3 we increase the bottleneck sizes to match the complexity of the HP environment. The full ablation study is reported in Appendix F.2.

In Fig. 2, we show 2D latent embeddings colored by task performance. We sample a uniform grid in the latent space and evaluate the reconstructed policies. The fully action-based baseline (**APC score & APC loss**) fails to recover a meaningful topological structure, resulting in an unstructured space dominated by low-performing behaviors. Evaluating the components in isolation reveals their individual limitations. Introducing only the occupancy-based objective (**APC score & OPC loss**) yields a smoother geometric gradient, but the lack of diverse high-performing policies in the APC-curated dataset restricts the presence of high-reward regions. Conversely, using the occupancy-based metric for curation with the action-based objective (**OPC score & APC loss**) successfully introduces high-reward policies into the latent space, but the action-matching loss scatters them erratically, failing to form cohesive clusters. The full occupancy-based pipeline (**OPC score & OPC loss**) effectively synergizes both components. The latent space is organized into a continuous, traversable geometry, with high-reward policies densely clustered in cohesive, structured regions.

### 5.3 Latent Behavior Optimization

**Q3:** On the role of OPC for latent optimization.




(a) MC, `standard`

(b) MC, `speed`

(c) MC, `height`

(d) RC, `speed`

(e) RC, `clockwise`

(f) RC, `radial`

(g) HP, `forward`

(h) HP, `backward`

(i) HP, `standstill`

(j) HP, `standard`

Figure 3: Performance comparison in MC, RC, and HP across different tasks. We report the average and 95% confidence interval over 10 runs of the best-performing algorithms for each task.

To answer **Q3**, we evaluate the downstream optimization performance. Following the experimental setting in Tenedini et al. (2026), we compare the performance of PGPE (Sehnke et al., 2008) operating in the parameter space, in the latent space derived from the APC pipeline (A-PGPE), and in the latent space derived from our OPC pipeline (O-PGPE) with various baselines, including DDPG (Lillicrap et al., 2016), TD3 (Fujimoto et al., 2018), and SAC (Haarnoja et al., 2018). We perform the optimization over all tasks and environments, and across several latent dimensions. The results can be observed in Fig. 3. For each task, we plot only the best curve for each family of algorithms: A-PGPE, O-PGPE, WS-O-PGPE, and the baselines. We also show the best-performing policy in the training dataset. The complete collection of training curves for all algorithms can be found in Appendix F.3.

As shown in Fig. 3, optimizing over the occupancy-matched latent space (O-PGPE) yields sample efficiency and final returns comparable to those of standard DRL baselines and action-matched manifolds (A-PGPE). Incorporating the Latent Warm Start (WS-O-PGPE) often accelerates the initial learning phase by initializing the agent in a high-reward region. For example, in MC, using Warm Start seemingly transforms the problem into a zero-shot optimization, superseding all other methods (Fig. 3a,3b), except for the task `height` (Fig. 3c).

The main advantage of our framework, however, emerges in complex environments, such as HP. In this domain, the action-matching proxy of APC fails to learn a behaviorally meaningful latent representation, leading to collapsed fine-tuning performance. Conversely, OPC successfully compresses high-performing behaviors in the latent manifold, which it later recovers to reliably combat standard baselines. Notably, in the complex locomotion task `backward` (Fig. 3h), OPC demonstrates an impressive capacity to generalize, discovering latent behaviors that achieve returns surpassing even the best-performing policies present in the training dataset. This task also highlights a trade-off of Latent Warm Start: coarse initialization can occasionally trap the optimization in local optima.

## 6 Conclusion

In this paper, we presented an enhanced framework for Latent Behavior Compression that fundamentally shifts the characterization of policy behaviors from immediate-action proxies to global state-visitation distributions. By combining an information-theoretic uniqueness metric for dataset curation with a differentiable mixture-occupancy matching objective, our pipeline successfully captures complex, long-horizon dynamics. Empirically, we demonstrated that it preserves critical outlier behaviors, constructs a behaviorally meaningful latent manifold, and enables highly sample-efficient adaptation. Crucially, optimizing within this occupancy-matched space enables agents to solve complex continuous control tasks that previous action-based compression methods fail to solve.

**Limitations and Future Directions.** While our framework establishes a robust foundation for state-occupancy manifold learning, its maximum asymptotic performance is ultimately limited by the behavioral diversity of the pre-training dataset. Future research could explore replacing random policy initialization with active, task-agnostic exploration frameworks, such as maximum-entropy or intrinsic-motivation algorithms (e.g., Eysenbach et al., 2018; Mutti et al., 2021). Furthermore, both the non-parametric $k$-NN estimator and the zeroth-order PGPE optimizer can exhibit high variance and fail to exploit the differentiable latent manifold. Consequently, employing variance-reduction techniques or exploring alternative latent optimization strategies beyond PGPE, such as Bayesian Optimization or gradient-aware methods like Latent-PPO (Han, 2026), could unlock even greater downstream adaptation capabilities within these highly compressed behavioral spaces.

### Acknowledgments

This work was supported by the Italian Ministry of University and Research (MUR) under the National Recovery and Resilience Plan (NRRP), and by the European Union (EU) under the NextGenerationEU project (CUP D43C24001670008).

## References


Joshua Achiam, Harrison Edwards, Dario Amodei, and Pieter Abbeel. Variational option discovery algorithms. *arXiv preprint arXiv:1807.10299*, 2018.

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C Courville, and Marc Bellemare. Reincarnating reinforcement learning: Reusing prior computation to accelerate progress. *Advances in Neural Information Processing Systems*, 35:28955–28971, 2022.

Jiří Ajgl and Miroslav Šimandl. Differential entropy estimation by particles. *IFAC Proceedings Volumes*, 44(1):11991–11996, 2011. ISSN 1474-6670. 18th IFAC World Congress.

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pp. 214–223. Pmlr, 2017.

Oscar Chang, Robert Kwiatkowski, Siyuan Chen, and Hod Lipson. Agent embeddings: A latent representation for pole-balancing networks. In *International Conference on Autonomous Agents and MultiAgent Systems*, pp. 656–664, 2019.

Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, 2016.

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.

Francesco Faccio, Vincent Herrmann, Aditya Ramesh, Louis Kirsch, and Jürgen Schmidhuber. Goal-conditioned generators of deep policies. In *AAAI Conference on Artificial Intelligence*, volume 37, pp. 7503–7511, 2023.

Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pp. 1587–1596, 2018.

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.

Karol Gregor, Danilo Rezende, and Daan Wierstra. Variational intrinsic control. *Workshop @ International Conference on Learning Representations*, 2017.

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.

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*, pp. 1861–1870, 2018.

Xiang Han. LG-H-PPO: offline hierarchical PPO for robot path planning on a latent graph. *Frontiers in Robotics and AI*, 12, 2026. ISSN 2296-9144.

Steven Hansen, Will Dabney, Andre Barreto, David Warde-Farley, Tom Van de Wiele, and Volodymyr Mnih. Fast task inference with variational intrinsic successor features. In *International Conference on Learning Representations*, 2019.

Shashank Hegde, Sumeet Batra, KR Zentner, and Gaurav Sukhatme. Generating behaviorally diverse policies with latent diffusion models. *Advances in Neural Information Processing Systems*, 36: 7541–7554, 2023.

Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. *Advances in neural information processing systems*, 29, 2016.

Sham M Kakade. A natural policy gradient. *Advances in Neural Information Processing Systems*, 14, 2001.

Věra Kůrková and Paul C. Kainen. Functionally equivalent feedforward neural networks. *Neural Computation*, 6(3):543–558, 1994.

Michael Laskin, Denis Yarats, Hao Liu, Kimin Lee, Albert Zhan, Kevin Lu, Catherine Cang, Lerrel Pinto, and Pieter Abbeel. Urlb: Unsupervised Reinforcement Learning benchmark. 2021.

Beiming Li, Sergio Rozada, and Alejandro Ribeiro. Learning policy representations for steerable behavior synthesis. *arXiv preprint arXiv:2601.22350*, 2026.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *International Conference on Learning Representations*, 2016.

Alberto Maria Metelli, Matteo Papini, Francesco Faccio, and Marcello Restelli. Policy optimization via importance sampling. *Advances in Neural Information Processing Systems*, 31, 2018.

Léo Meynent, Ivan Melev, Konstantin Schürholt, Goeran Kauermann, and Damian Borth. Structure is not enough: Leveraging behavior for neural network weight reconstruction. *Workshop @ International Conference on Learning Representations*, 2025.

Atsushi Miyamae, Yuichi Nagata, Isao Ono, and Shigenobu Kobayashi. Natural policy gradient methods with parameter-based exploration for control tasks. *Advances in Neural Information Processing Systems*, 23, 2010.

Alessandro Montenegro, Marco Mussi, Alberto Maria Metelli, and Matteo Papini. Learning optimal deterministic policies with stochastic policy gradients. In *International Conference on Machine Learning*, pp. 36160–36211, 2024.

Andrew William Moore. Efficient memory-based learning for robot control. Technical report, University of Cambridge, 1990.

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, 2021. DOI: 10.1609/aaai.v35i10.17091.

Mirco Mutti, Stefano Del Col, and Marcello Restelli. Reward-free policy space compression for reinforcement learning. In *International Conference on Artificial Intelligence and Statistics*, pp. 3187–3203. PMLR, 2022.

William Peebles, Ilija Radosavovic, Tim Brooks, Alexei A Efros, and Jitendra Malik. Learning to learn with generative models of neural network checkpoints. *arXiv preprint arXiv:2209.12892*, 2022.

Jan Peters and Stefan Schaal. Reinforcement learning of motor skills with policy gradients. *Neural networks*, 21(4):682–697, 2008.

Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

Nemanja Rakicevic, Antoine Cully, and Petar Kormushev. Policy manifold search: Exploring the manifold hypothesis for diversity-based neuroevolution. In *Genetic and Evolutionary Computation Conference*, pp. 901–909, 2021.

Thomas Rückstiess, Frank Sehnke, Tom Schaul, Daan Wierstra, Yi Sun, and Jürgen Schmidhuber. Exploring parameter space in reinforcement learning. *Paladyn*, 1(1):14–24, 2010.

Andrei A. Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation, 2016.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Konstantin Schürholt, Dimche Kostadinov, and Damian Borth. Self-supervised representation learning on neural network weights for model characteristic prediction. *Advances in Neural Information Processing Systems*, 34:16481–16493, 2021.

Konstantin Schürholt, Boris Knyazev, Xavier Giró-i Nieto, and Damian Borth. Hyper-representations as generative models: Sampling unseen neural network weights. *Advances in Neural Information Processing Systems*, 35:27906–27920, 2022.

Konstantin Schürholt, Michael W Mahoney, and Damian Borth. Towards scalable and versatile weight space learning. In *International Conference on Machine Learning*, pp. 43947–43966, 2024.

Frank Sehnke, Christian Osendorfer, Thomas Rückstieß, Alex Graves, Jan Peters, and Jürgen Schmidhuber. Policy gradients with parameter-based exploration for control. In *International Conference on Artificial Neural Networks*, pp. 387–396. Springer, 2008.

Davide Tenedini, Riccardo Zamboni, Mirco Mutti, and Marcello Restelli. From parameters to behaviors: Unsupervised compression of the policy space. In *International Conference on Learning Representations*, 2026.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *International Conference on Intelligent Robots and Systems*, pp. 5026–5033, 2012.

Riccardo Zamboni, Mirco Mutti, and Marcello Restelli. Towards principled unsupervised multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 2025.

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)*, pp. 1165–1170, 2020. DOI: 10.1109/DDCLS49620.2020.9275169.

# Supplementary Materials

*The following content was not necessarily subject to peer review.*

## A Extended Related Work

Our framework sits at the intersection of unsupervised behavior discovery, weight-space learning, and occupancy matching, thereby unifying these domains to learn a functionally grounded, low-dimensional policy manifold.

**Unsupervised RL and Behavior Discovery.** A key challenge in Deep Reinforcement Learning is the dependency on extrinsic rewards, which limits an agent's ability to generalize. To address this, Unsupervised RL (URL) methods seek to learn useful behaviors before a specific goal is defined. Information-theoretic frameworks like DIAYN (Eysenbach et al., 2018), as well as other skill-discovery methods (Gregor et al., 2017; Achiam et al., 2018; Hansen et al., 2019), aim to discover diverse skills by maximizing the mutual information between states and latent variables. Similarly, MEPOL (Mutti et al., 2021) utilizes a maximum-entropy objective to drive an agent toward maximal state-space coverage, while more recently, TRPE (Zamboni et al., 2025) utilizes a similar approach to enforce behavioral diversity in multi-agent systems. While these methods successfully identify robust task-agnostic initializations, they typically revert to operating in the original high-dimensional parameter space once supervised fine-tuning begins. Our framework, instead, identifies a continuous global behavioral manifold that inherently facilitates and accelerates this downstream adaptation process.

**Policy Compression and Weight Space Learning.** The goal of learning a latent representation of neural network parameters connects our approach to the field of Weight Space Learning (WSL). In RL, recent generative models have explored task-specific parameter generation conditioned on goals (Faccio et al., 2023) or performance checkpoints (Peebles et al., 2022). In supervised learning, *hyper-representations* (Schürholt et al., 2021; 2022; 2024) aim to build task-agnostic embeddings from a "zoo" of trained models. However, standard WSL methods and traditional policy compression techniques (Rusu et al., 2016) face a critical hurdle: the vast number of parameter-space symmetries (e.g., neuron permutations and scaling) inherent in neural architectures (Kůrková & Kainen, 1994). To circumvent this, recent works in supervised learning supplement standard parameter reconstruction losses with behavioral output matching (Meynent et al., 2025). In Unsupervised Reinforcement Learning, other approaches tackle policy space simplification mathematically, such as reducing the cardinality of the policy space (Mutti et al., 2022). However, these strict constraints often lead to NP-hard optimization problems. Our approach sidesteps permutation symmetries entirely. Building on the Latent Behavior Compression framework (Tenedini et al., 2026), we use a functional compression metric rather than parameter reconstruction, thereby allowing the generative model to naturally assimilate mathematically divergent yet functionally identical parameterizations.

**Policy Manifolds and Quality Diversity.** The existence of low-dimensional manifolds embedded within policy parameter spaces has been hypothesized (Rakicevic et al., 2021) and explored using Variational Autoencoders (VAEs) to analyze pre-trained expert embeddings (Chang et al., 2019). Similar architectures have been applied in Quality Diversity to improve the sample efficiency of diversity-based search (Rakicevic et al., 2021) or to distill large policy archives into generative models (Hegde et al., 2023). Notably, these methods rely on *parameter-reconstruction losses*, which fundamentally restrict their compression ratios (e.g., up to $19 : 1$ in Hegde et al. (2023)). By optimizing a behavioral objective directly, our unsupervised pipeline achieves substantially greater compression while providing a topologically smooth space for downstream optimization.

**Occupancy Matching and Policy Optimization.** Comparing behaviors formally often involves analyzing state occupancy distributions, a concept recently utilized to learn steerable latent policy representations via expected state-action features (Li et al., 2026). Comparing behaviors is also the foundation of Imitation Learning (IL), where methods like GAIL (Ho & Ermon, 2016) and Wasserstein-GANs (Goodfellow et al., 2020; Arjovsky et al., 2017; Gulrajani et al., 2017; Zhang

et al., 2020) minimize the divergence between an agent's behavior and a target expert distribution. Our contribution scales these concepts from single-expert matching to population-level alignment by developing a mixture-occupancy matching objective. Once this manifold is learned, we optimize downstream tasks using Policy Gradient with Parameter-based Exploration (PGPE) (Sehnke et al., 2008; Rückstiess et al., 2010; Miyamae et al., 2010; Montenegro et al., 2024). While zeroth-order methods like PGPE and first-order methods (Peters & Schaal, 2008; Lillicrap et al., 2016; Kakade, 2001; Schulman et al., 2017) traditionally struggle to scale in highly redundant parameter spaces, executing PGPE within our highly compressed, behaviorally organized latent space resolves these scalability bottlenecks, avoiding the complex decoder Jacobian calculations required by prior manifold optimization approaches (Rakicevic et al., 2021).

# B Policy Dataset Curation: Derivations and Estimators

This section provides the formal derivations and practical estimation details for the information-theoretic uniqueness score introduced in Section 4.

## B.1 Derivation of the Uniqueness Score

Let $M = |\Psi|$ denote the number of policies in the population. Let $d_i(s)$ for $i = 1, \dots, M$ be the state occupancy distribution induced by the $i$-th policy $\pi_i \in \Psi$. We consider the uniform mixture over these occupancy distributions by assuming that each policy is selected with equal probability, $p(i) = \frac{1}{M}$.

This induces a joint distribution over states and policies given by $p(s, i) = p(s \mid i)p(i) = \frac{1}{M} d_i(s)$, from which the marginal state distribution, corresponding to the mixture occupancy $d_\Psi$, is obtained as $d_\Psi(s) = \sum_{i=1}^{M} p(s, i) = \frac{1}{M} \sum_{i=1}^{M} d_i(s)$.

Let $H(X)$ denote the differential entropy of a random variable $X$. Starting from the chain rule $H(X, Y) = H(X) + H(Y \mid X)$ and using the symmetry of joint differential entropy, we can write:

$$\begin{aligned} H(X) + H(Y \mid X) &= H(Y) + H(X \mid Y) \\ H(X) &= H(X \mid Y) + MI(X, Y), \end{aligned}$$

where $MI(X, Y)$ denotes the mutual information. We now apply this to the random variables $S \sim d_\Psi(s)$ and $\pi \sim p(i)$. The conditional entropy becomes:

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

The mutual information can be expanded as:

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

Combining the two terms, we obtain the entropy decomposition of the mixture occupancy distribution:

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

The term inside the summation represents the marginal contribution of the $i$-th policy to the total entropy of the mixture, which we utilize as our uniqueness score $\rho(\boldsymbol{\theta}_i)$.

### B.2 Tractable Estimation via GMMs

Because the analytical form of the occupancy $d_{\boldsymbol{\theta}_i}$ is unknown, we estimate the scores using trajectory data. We first fit Gaussian Mixture Models (GMMs) to the temporally downsampled trajectories of each policy, yielding the continuous density estimators $\hat{d}_{\boldsymbol{\theta}_i}(s)$ for individual policies and $\hat{d}_\Psi(s)$ for the global mixture.

Using a set of $N$ particles $\mathcal{D}_{\boldsymbol{\theta}_i} = \{s_j\}_{j=1}^{N}$ sampled from the individual density estimate $\hat{d}_{\boldsymbol{\theta}_i}$, we apply Monte Carlo (MC) integration to approximate the values of the differential entropy and the KL divergence:

$$
\begin{aligned}
\widehat{H}(d_{\boldsymbol{\theta}_i}) &= -\frac{1}{N} \sum_{j=1}^{N} \log \hat{d}_{\boldsymbol{\theta}_i}(s_j), \\
\widehat{D_{\mathrm{KL}}}(d_{\boldsymbol{\theta}_i} \parallel d_\Psi) &= \frac{1}{N} \sum_{j=1}^{N} \left[ \log \hat{d}_{\boldsymbol{\theta}_i}(s_j) - \log \hat{d}_\Psi(s_j) \right].
\end{aligned}
$$

### B.3 KL Divergence Bound and the Saturation Effect

When calculating $\widehat{D_{\mathrm{KL}}}$, the estimation of the mixture occupancy distribution $\hat{d}_\Psi$ must be handled carefully to avoid a *saturation effect* that degrades the discriminative power of the scores as the population grows. This is due to a strict upper bound on the KL divergence between a single component and its mixture.

**Proposition 1.** *Let $m(x) = \sum_{j=1}^{M} w_j p_j(x)$ be a mixture of $M$ probability density functions with weights $w_j \geq 0$ and $\sum w_j = 1$. For any component $i$, the KL divergence is bounded: $D_{\mathrm{KL}}(p_i \parallel m) \leq -\log w_i$.*

*Proof.* By the definition of the mixture, $m(x) = w_i p_i(x) + \sum_{j \neq i} w_j p_j(x)$. Because density functions and weights are non-negative, $m(x) \geq w_i p_i(x)$, which implies $\frac{p_i(x)}{m(x)} \leq \frac{1}{w_i}$.

Since the logarithm is monotonically increasing, $\log \frac{p_i(x)}{m(x)} \leq -\log w_i$. The KL divergence is the expectation of this log-ratio over $p_i$:

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

Since $-\log w_i$ is a constant and $p_i(x)$ integrates to 1, $D_{\mathrm{KL}}(p_i \parallel m) \leq -\log w_i$. ■

In our uniform mixture, $w_i = 1/M$, yielding the bound $D_{\mathrm{KL}}(d_{\boldsymbol{\theta}_i} \parallel d_\Psi) \leq \log M$. If $\hat{d}_\Psi$ is estimated using exhaustive aggregation across a massive population $\Psi$, the KL term mathematically vanishes relative to the scale of the state space, failing to capture uniqueness. To preserve metric sensitivity, we restrict the calculation of the global reference $\hat{d}_\Psi$ to a representative subset of the population.

## C Numerically Stable Importance Weight Computation

As introduced in Section 4, the computation of the importance weights follows the density-ratio methodology of Mutti et al. (2021), adapted here to accommodate mixtures of policies. We associate each particle $s_t^j$ with the importance weight $w_j$ calculated for the full trajectory $\tau_j$ that produced it.

Because the non-normalized weight $\overline{w}_j$ involves the product of numerous probability values lying in the interval $[0, 1]$, both the numerator and denominator can quickly vanish to zero over long horizons. To prevent arithmetic underflow and a loss of numerical precision, we transform the product of probabilities into a sum of log-probabilities and compute the weights entirely in log-space using the *Log-Sum-Exp* (LSE) trick.

Let $\ell_{i,j}$ be the log-probability of trajectory $\tau_j$ under the $i$-th policy of the original dataset mixture $\pi_{\boldsymbol{\theta}_i} \in \mathcal{D}_\Theta$:

$$\ell_{i,j} = \log p(\tau_j \mid \pi_{\boldsymbol{\theta}_i}) = \sum_{t=0}^{T} \log \pi_{\boldsymbol{\theta}_i}(a_t^j \mid s_t^j).$$

Similarly, let $\ell'_{i,j}$ be the log-probability of the same trajectory under the $i$-th reconstructed target policy $\pi_{\hat{\boldsymbol{\theta}}_i} \in \mathcal{D}_{\hat{\Theta}}$. The non-normalized weight $\overline{w}_j$ is the ratio of these two sums of exponentials. In log-space, letting $\lambda_j = \log \overline{w}_j$, this is expressed as:

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

To prevent overflow or underflow during these summations, each term is computed using the LSE function:

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

By extracting the maximum value $c$, we ensure that the largest exponentiated term inside the summation is strictly $\exp(0) = 1$, safely shifting the range of the calculation into a stable floating-point region. Thus, the non-normalized log-weight is robustly computed as:

$$\lambda_j = \mathrm{LSE}_{i \in \mathcal{D}_{\hat{\Theta}}}(\ell'_{i,j}) - \mathrm{LSE}_{i \in \mathcal{D}_\Theta}(\ell_{i,j}).$$

The final step is self-normalization, which ensures that the weights over the dataset sum to unity, $w_j = \overline{w}_j / \sum_k \overline{w}_k$. To maintain numerical stability until the very end of the computation, we perform this normalization in the log-domain as well:

$$w_j = \exp\left(\lambda_j - \mathrm{LSE}_k(\lambda_k)\right).$$

This approach guarantees that we exponentiate only the final output, preserving high numerical precision throughout the density-ratio estimation process.

**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 $\alpha$.
**Output:** Generative behavioral mapping $g_{\boldsymbol{\zeta}}$.
Randomly initialize autoencoder parameters: encoder $\boldsymbol{\xi}$ and decoder $\boldsymbol{\zeta}$.
**repeat**
  Sample a mini-batch of $P$ policies $\psi = \{\boldsymbol{\theta}_i\}_{i=1}^{P} \subset \mathcal{D}_\Theta$
  Retrieve corresponding trajectories $\mathcal{D}_\psi \subset \mathcal{D}_\tau$
  **for** $i = 1$ **to** $I$ **do**
    Reconstruct parameters: $\hat{\psi} = \{\hat{\boldsymbol{\theta}}_i = g_{\boldsymbol{\zeta}}(f_{\boldsymbol{\xi}}(\boldsymbol{\theta}_i)) \mid \boldsymbol{\theta}_i \in \psi\}$
    Compute mixture-matching loss: $\mathcal{L}_B(\boldsymbol{\xi}, \boldsymbol{\zeta}) = D_{\mathrm{KL}}\Big(d_\psi \parallel d_{\hat{\psi}}\Big)$ {via Eq. 1}
    Update weights: $\{\boldsymbol{\xi}, \boldsymbol{\zeta}\} \leftarrow \{\boldsymbol{\xi}, \boldsymbol{\zeta}\} - \alpha \nabla_{\{\boldsymbol{\xi}, \boldsymbol{\zeta}\}} \mathcal{L}_B(\boldsymbol{\xi}, \boldsymbol{\zeta})$
  **end for**
**until** convergence or full population coverage
**return** $g_{\boldsymbol{\zeta}}$

# D Autoencoder Training Pipeline

Optimizing the mixture-occupancy matching objective over the entire policy dataset $\mathcal{D}_\Theta$ at once is computationally prohibitive. To scale the training process, we employ a mini-batch stochastic gradient descent strategy over the policy population.

During each training step, we uniformly sample a subset $\psi$ of $P$ policies from the dataset, and fetch their corresponding pre-collected trajectories $\mathcal{D}_\psi \subset \mathcal{D}_\tau$. A major advantage of our non-parametric importance-sampling formulation is the ability to reuse the exact trajectory batch across multiple sequential parameter updates, thereby drastically increasing sample efficiency. Specifically, for each sampled batch $\psi$, we execute an inner loop consisting of $I$ consecutive gradient descent steps of the autoencoder's parameters $\{\boldsymbol{\xi}, \boldsymbol{\zeta}\}$.

Critically, because the decoder's weights (and consequently, the reconstructed policies $\hat{\psi}$) evolve after each step, the true density ratio shifts. Therefore, the importance weights are re-evaluated at every step using the numerically stable Log-Sum-Exp procedure described in Appendix C. This nested update scheme significantly accelerates convergence and improves sample efficiency. Training iterations are repeated until the population has been sufficiently sampled or the autoencoder has converged. The complete training procedure is summarized in Algorithm 1.

**Practical Implementation of Policy Stochasticity.** While our theoretical framework models policies as strictly stochastic ($\pi \in \Delta(\mathcal{A})$) to accommodate density ratio estimation, evaluating highly exploratory, randomly initialized continuous policies can yield erratic, noisy actions due to untrained variance parameters. To ensure meaningful and stable state-space coverage during the Policy Dataset Curation phase, and during downstream PGPE optimization, we deploy the deterministic version of the policies (i.e., we execute only the mean of the action distribution). Consequently, during the autoencoder training phase, we evaluate the stochastic likelihood of these deterministically collected trajectories. We acknowledge that evaluating stochastic probabilities on deterministically generated data introduces bias into the importance-sampling estimator. However, this is a necessary and highly effective empirical trade-off: it strips catastrophic noise from the behavioral rollouts while utilizing the variance parameters as a continuous smoothing mechanism. This guarantees a strictly positive probability density during training, circumventing the vanishing denominators and intractable density ratios that would occur if purely deterministic Dirac-delta distributions were used.

## E Experimental Settings

This section provides the details regarding the architecture, environment, and training procedure used to conduct the experiments presented in Section 5.

### E.1 Architecture and Training

**Policy Networks.** To model the behaviors, we use fully-connected, feed-forward Multi-Layer Perceptrons (MLPs). The input layer has $|\mathcal{S}|$ neurons and is preceded by a normalization layer that standardizes the state features to have zero mean and unit variance. The policies consist of two hidden layers of 32 neurons, each followed by Exponential Linear Unit (ELU) activations. The final output layer consists of $|\mathcal{A}|$ neurons with a `tanh` activation to squash the actions into the valid environmental bounds. The policies have roughly $10^3$ parameters: $1,218$ for MC, $1,412$ for RC, and $1,638$ for HP. The weights are sampled from independent uniform distributions $\boldsymbol{\theta}_i \sim U(-2.5, 2.5)^n$, where $n$ is the number of parameters.

**Autoencoder Architecture.** We utilize a fixed, symmetric autoencoder architecture to construct the behavioral manifold. First, the weights are standardized to have zero mean and unit variance. Then, the encoder projects the $n$-dimensional policy parameter vector through two fully connected hidden layers with 25 and 10 neurons, respectively, into a $k$-dimensional bottleneck. The decoder symmetrically maps the $k$-dimensional latent code back through layers of 10 and 25 neurons to reconstruct the $n$-dimensional weight vector. The encoder and decoder output layers are not activated. All other layers utilize ELU activation functions. Based on the environment, the bottleneck can have shape $k \in \{1, 2, 3, 5, 8\}$.

**Autoencoder Training.** During the Latent Behavior Compression phase, the autoencoder is trained on a curated dataset of $|\mathcal{D}_\Theta| = 50,000$ policies. The dataset is then reduced to the top 5%. To compute the mixture-matching loss, we sample 20 trajectories per policy, which we found to be an optimal trade-off between estimator variance and computational efficiency. For the importance-sampling estimator, the nearest-neighbors parameter is fixed at $k = 30$ across all domains. We train with a small batch size of 5; while this introduces gradient noise, it crucially prevents the model from collapsing into a trivial mean-policy solution, thereby forcing the manifold to capture distinct, high-reward behavioral modes that would otherwise be smoothed out. The total number of training iterations is computed to ensure that each policy in the dataset is sampled at least once with probability greater than $0.99$. Furthermore, policies are sampled uniformly at random rather than sequentially to prevent catastrophic forgetting.

### E.2 Environments

We evaluate our framework on three continuous control environments from the Gymnasium library. For each environment, the unsupervised pre-training phase is entirely reward-free, while the downstream optimization phase evaluates few-shot adaptation to four distinct reward functions.

**Mountain Car Continuous (MC).** In this environment, a car is stochastically placed in a sinusoidal valley and must build momentum to traverse the hills. The state space $\mathcal{S} \subseteq \mathbb{R}^2$ captures position $p \in [-1.2, 0.6]$ and velocity $v \in [-0.07, 0.07]$, while the action space $\mathcal{A} \subseteq \mathbb{R}^1$ represents the applied acceleration. We define four tasks: (1) **`standard`**: a sparse $+100$ reward for reaching the right goal ($p \geq 0.45$), with a $-0.1a^2$ control penalty; (2) **`left`**: identical to **`standard`**, but the goal is at the left peak ($p \leq -1.1$); (3) **`height`**: a dense reward $R_t = h^2$ for height $h \geq 0.2$; and (4) **`speed`**: a dense reward $R_t = v^2$. Episodes terminate upon reaching the standard goal or after 999 steps.

**Reacher (RC).** RC features a two-jointed robotic arm moving in a 2D plane. To focus strictly on task-agnostic behavioral discovery, we remove all target-related information from the observations. The normalized state space $\mathcal{S} \subseteq \mathbb{R}^6$ captures the sines, cosines, and angular velocities of the two joints, and the action space $\mathcal{A} \subseteq \mathbb{R}^2$ controls the hinge torques. For the purpose of normalization, we consider the state bounded between the vectors $[-1, -1, -1, -1, -5, -5]$ and $[1, 1, 1, 1, 5, 5]$. We

define four binary tasks evaluated on the fingertip's kinematics ($R_t = 1$ if met, else 0): (1) **speed**: linear velocity $> 6$; (2) **clockwise**: tangential velocity $< -1$; (3) **c-clockwise**: tangential velocity $> 1$; and (4) **radial**: radial velocity $> 3$. Episodes terminate after 50 steps.

**Hopper (HP).** HP is a complex planar monopod robot governed by a normalized 11-dimensional state space capturing height, joint angles, and velocities, controlled via a 3-dimensional action space governing the hip, knee, and ankle torques. For the purpose of normalization, we consider the state bounded between the vectors $[0.7, -0.2, -2.7, -2.7, -0.8, -5.0, -5.0, -5.0, -5.0, -5.0, -5.0]$ and $[1.5, 0.2, 0.0, 0.0, 0.8, 5.0, 5.0, 5.0, 5.0, 5.0, 5.0]$. We define four locomotion tasks: (1) **forward**: $x$-axis velocity $> 1$; (2) **backward**: $x$-axis velocity $< -1$; (3) **standstill**: $x$-axis velocity within $[-0.05, 0.05]$; and (4) **standard**: the default Gymnasium composite reward consisting of a $+1$ healthy survival bonus, a forward velocity term, and a control penalty. The first three tasks use binary rewards. Episodes terminate if the robot falls or after 1,000 steps.

### E.3 Hyperparameters

**Baselines.** We employ the baseline algorithms (PPO, DDPG, and SAC) from the **StableBaselines3** (Raffin et al., 2021) library, using the same hyperparameters as in Tenedini et al. (2026) (for MC and RC), specifically:

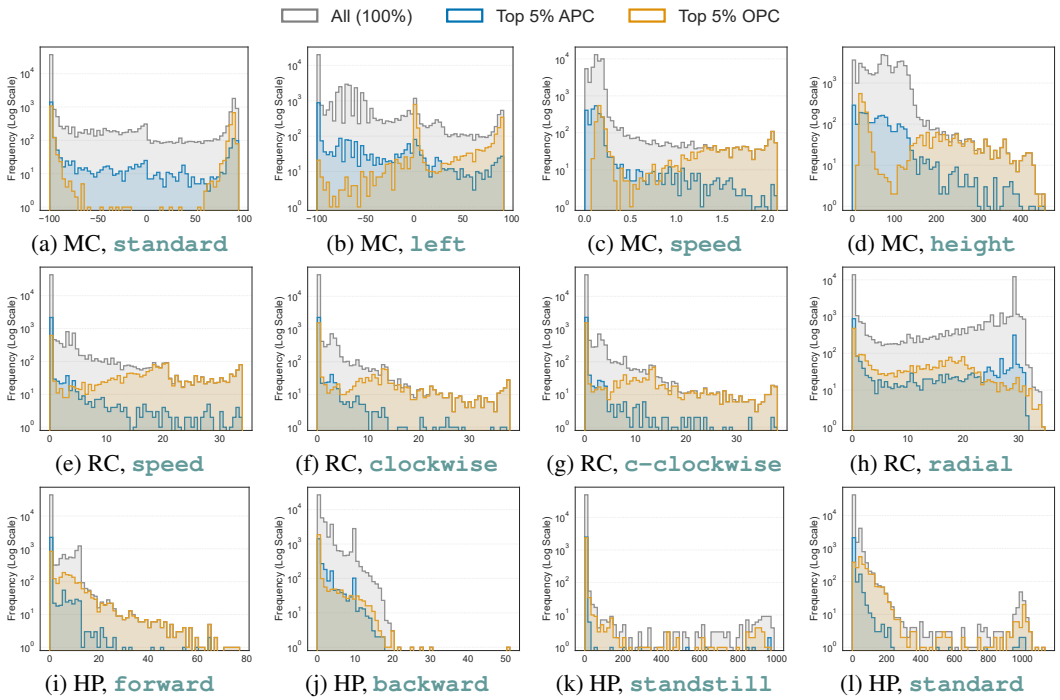
- **Mountain Car:**
  - **DDPG:** Exploration noise standard deviation of 0.75 and 0.65, and a replay buffer size of 50,000.
  - **SAC:** Generalized State Dependent Exploration (gSDE), entropy coefficient of 0.1, soft update coefficient of 0.01, train frequency of 32, and 32 gradient steps per rollout.
  - **PPO:** Generalized State Dependent Exploration (gSDE), learning rate of 0.0001, batch size of 256, and 4 optimization epochs per rollout.
- **Reacher:** Hyperparameters are identical to those used in Mountain Car, with the exception of the DDPG exploration noise standard deviation, which is set to 0.5.

Where not specified, including for all the algorithms in the HP environment, the hyperparameters are set to the already-fine-tuned values from the RL Baselines3 Zoo.

**PGPE.** Latent Behavior Optimization is performed using Policy Gradients with Parameter-based Exploration (PGPE), optimized via Adam. We maintain a population size of 8 across all environments and use an exponential scheduler for learning-rate decay. To prevent overshooting high-reward regions and becoming trapped in zero-gradient plateaus, we reduce the first moment decay to $\beta_1 = 0.1$. Hyperparameters are tailored to specific environment objectives: for the **Mountain Car** (MC) **standard** and **left** rewards, we set the center and standard deviation learning rates to 0.11 and 0.1, respectively (initial $\sigma = 1.0$). The **speed** task uses more conservative rates (0.02 center, 0.01 std), while the **height** task requires a center rate of 0.01 and a reduced initial $\sigma$ of 0.1. For **Reacher** (RC), we utilize a center learning rate of 0.1 and a standard deviation learning rate of 0.05, whereas Hopper (HP) is optimized with a center learning rate of 0.11.

For the **Warm Start** variation of PGPE, we adopt the hyperparameters previously used for Reacher in the Hopper environment (center LR $= 0.1$, std LR $= 0.05$). In contrast, for Mountain Car, we employ a more conservative configuration: a center learning rate of 0.01, an initial standard deviation of 0.1, and a standard deviation learning rate of 0.05. This choice is informed by the observation that initial random sampling often identifies high-reward regions early, necessitating only marginal refinement rather than extensive further optimization. For the warm-start initialization, we sample an initial set of policies with size proportional to the latent space dimensionality: 40, 56, and 80 policies for the 3D, 5D, and 8D spaces, respectively.

**Reproducibility.** All experimental components are fixed with specific random seeds to ensure reproducibility and facilitate fair comparisons. The seeding strategy for the pipeline is as follows:




(a) MC, `standard` (b) MC, `left` (c) MC, `speed` (d) MC, `height`

(e) RC, `speed` (f) RC, `clockwise` (g) RC, `c-clockwise` (h) RC, `radial`

(i) HP, `forward` (j) HP, `backward` (k) HP, `standstill` (l) HP, `standard`

Figure 4: Reward distributions comparison in MC, RC, and HP.

- **Dataset Generation and Curation:** A constant seed of $0$ is applied across all stages, including policy generation, particle collection, and score computation.
- **Latent Behavior Compression:** For the main optimization experiments, ten distinct latent spaces are trained using seeds $0, \ldots, 9$. The ablation studies that compare latent spaces specifically utilize seeds $888$ and $999$.
- **Latent Behavior Optimization:** The PGPE optimization scripts are initialized with seed $0$, while the baselines are evaluated across ten independent runs each using seeds $0, \ldots, 9$.

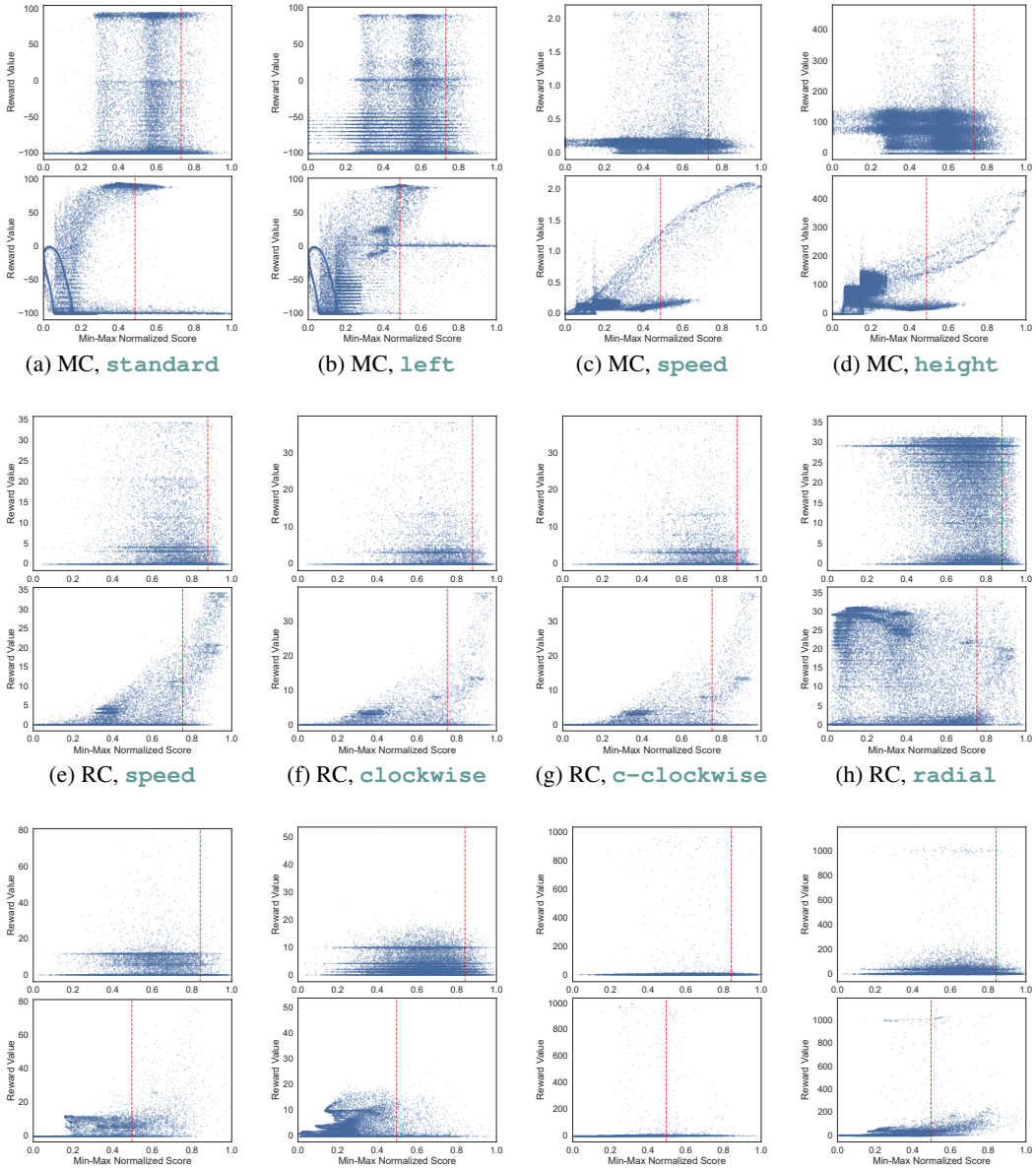
# F Additional Experiments

In this section, we present the full list of experiments presented in Section 5, including those that could not fit in the main text.

## F.1 Policy Dataset Curation

In this experiment, we generate a dataset of $50,000$ policies for each environment (MC, RC, and HP). We then apply a top $5\%$ threshold based on scores generated by both the APC pipeline (obtained via novelty search in the Euclidean space of immediate actions) and the OPC pipeline (obtained via our information-theoretic uniqueness metric in the occupancy space). In Fig. 4, we visualize both dataset distributions compared to the complete ($100\%$) dataset. Consequently, in Fig. 5, we compare the distribution of scores that led to the previously mentioned thresholding.

As noted in Section 5.1, the scores generated by the APC pipeline tend to have less structure, consequently leading to the loss of rare, high-performing behaviors in the thresholded dataset. Conversely, the OPC scores generally favor unique behaviors. Interestingly, for tasks in which achieving a good performance is trivial, the scores successfully mitigate both the probability mass of low-performing and high-performing policies introduced by the random initialization (e.g., the `radial` task for RC in Fig. 4h).




(a) MC, `standard` (b) MC, `left` (c) MC, `speed` (d) MC, `height`

(e) RC, `speed` (f) RC, `clockwise` (g) RC, `c-clockwise` (h) RC, `radial`

(i) HP, `forward` (j) HP, `backward` (k) HP, `standstill` (l) HP, `standard`

Figure 5: Correlation between APC (top row) and OPC (bottom row) scores and rewards in MC, RC, and HP. The vertical dashed line indicates the top 5% selection threshold.

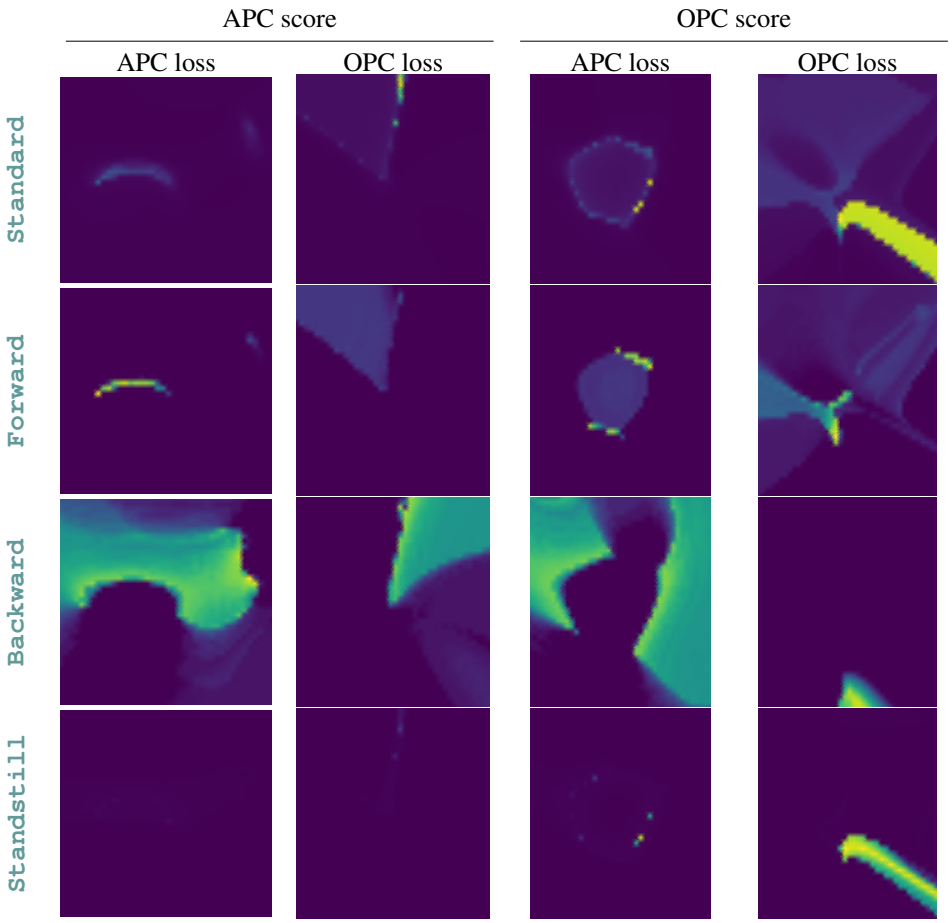



Figure 6: HP, seed 888, 2D

### F.2 Latent Behavior Compression

In this ablation study, we first generate a dataset of $50,000$ policies for the HP environment. We then curate it to 5% using both the scores from the OPC and APC pipelines. Finally, we train two autoencoders: one with the action-matching objective from the APC pipeline and one with the occupancy-matching objective from the OPC pipeline. In all, we obtain four latent spaces: one obtained by running the full APC pipeline; one obtained by running the full OPC pipeline; one obtained by optimizing the OPC objective on the APC curated dataset; and, finally, one obtained by optimizing the APC objective on the OPC curated dataset. We repeat this process with two different latent dimensions: 2D and 3D. We compare the obtained topologies in figures 6, 7, 8 and 9.

### F.3 Latent Behavior Optimization

In this experiment, we compare the optimization performances of our pipeline (O-PGPE) against similar methods and other competitive DRL baselines, which include: P-PGPE, i.e., standard PGPE running in the parameter space, SAC, DDPG, and TD3, all utilizing policies of the same shape as the one being compressed by our autoencoder. Additionally, we compare our results with A-PGPE (Latent PGPE on the Action-based Policy Compression manifolds). Both O-PGPE and A-PGPE are tested on the same bottleneck dimensions, namely: $k = 1, 2, 3$ for MC, and $k = 3, 5, 8$ for RC and HP. Finally, we evaluate the performance of WS-O-PGPE (Warm Start O-PGPE) with bottleneck size $k = 3$, 5, and 8.

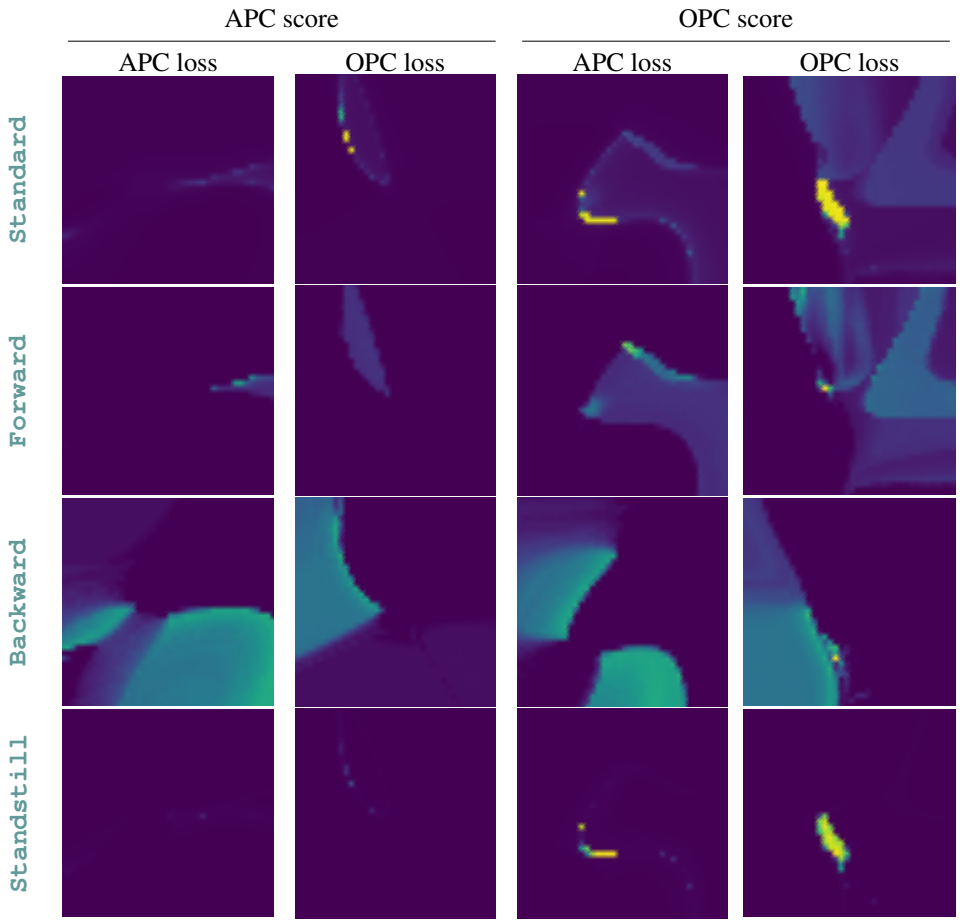



Figure 7: HP, seed 999, 2D

We present the results in categories: Occupancy-based Policy Compression (Fig. 10), Action-based Policy Compression (Fig. 11), and baselines (Fig. 12).

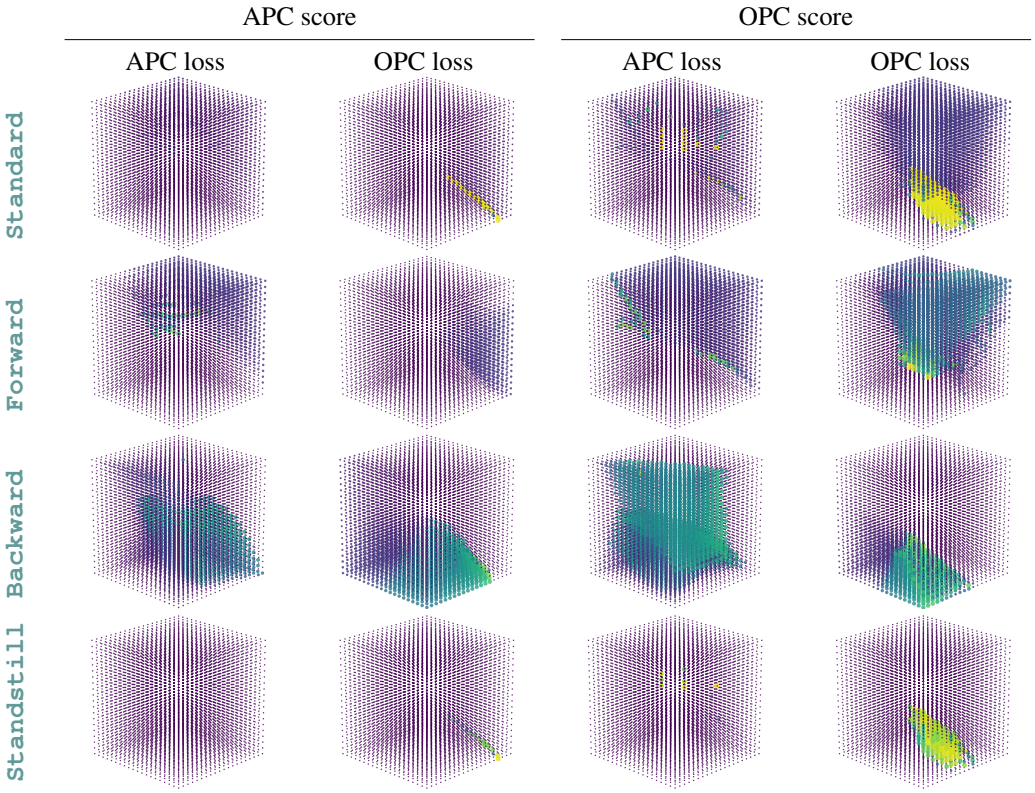



Figure 8: HP, seed 888, 3D

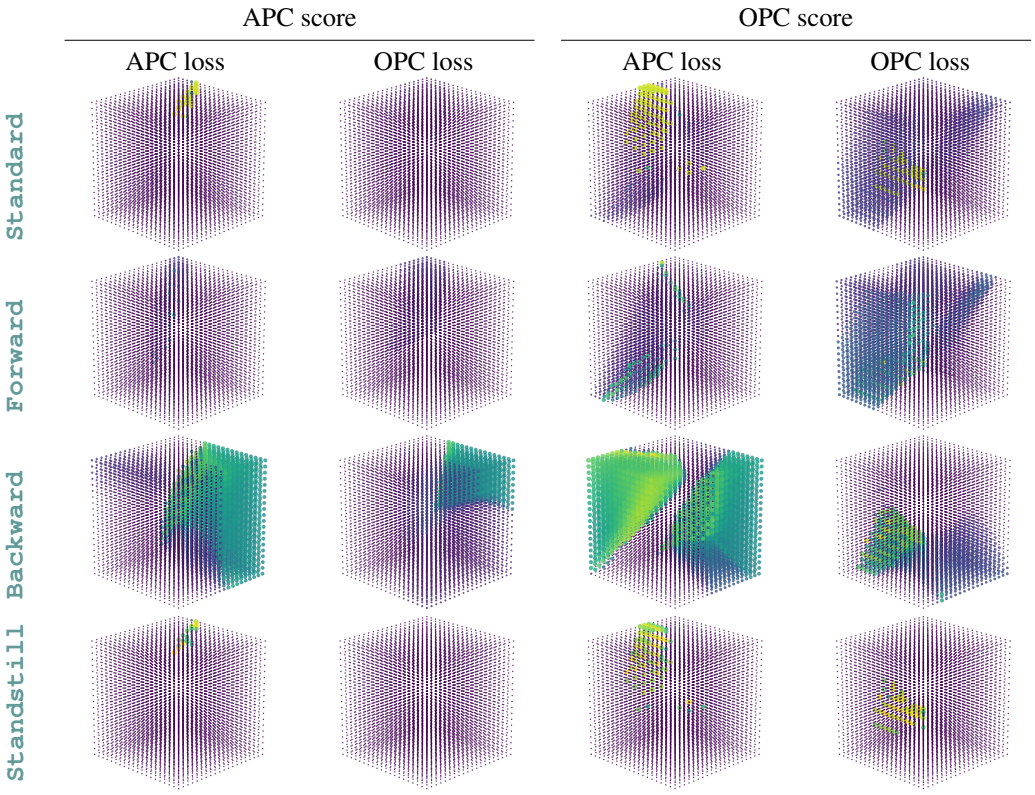
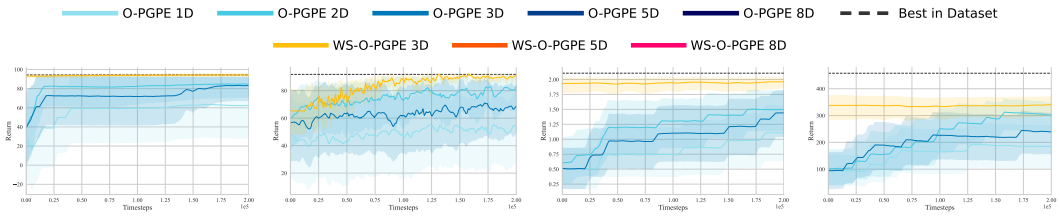
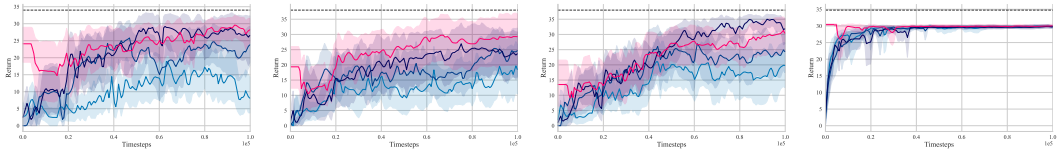


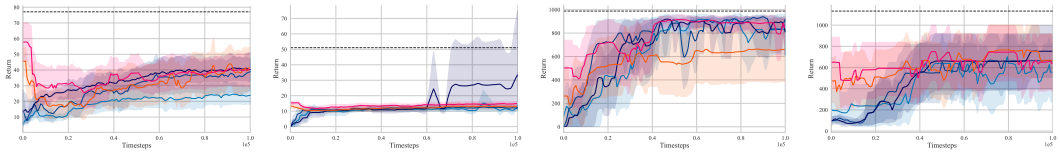

Figure 9: HP, seed 999, 3D




(a) MC, `standard` (b) MC, `left` (c) MC, `speed` (d) MC, `height`

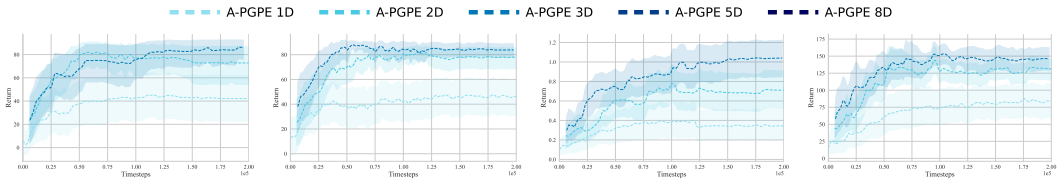


(e) RC, `speed` (f) RC, `clockwise` (g) RC, `c-clockwise` (h) RC, `radial`

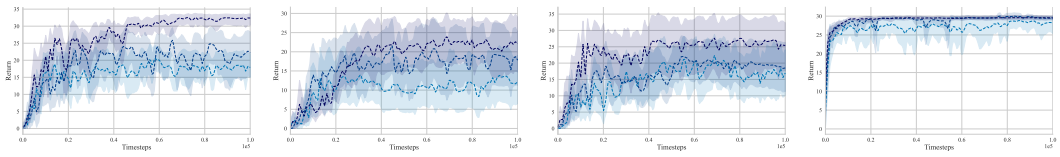


(i) HP, `forward` (j) HP, `backward` (k) HP, `standstill` (l) HP, `standard`

Figure 10: Performance comparison in MC, RC, and HP for different tasks using **Occupancy-based Policy Compression**.

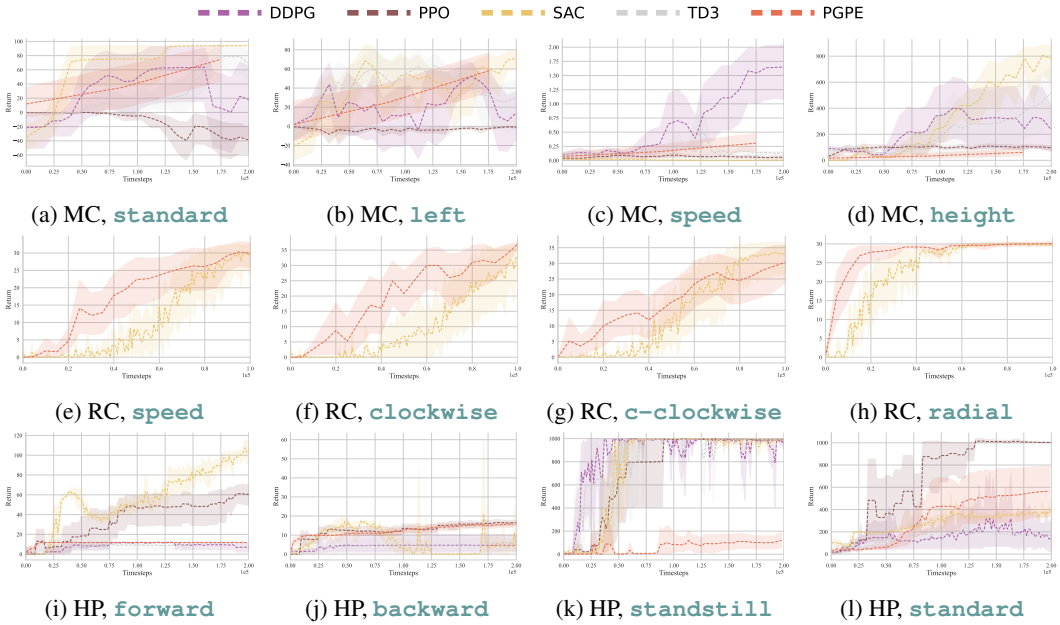



(a) MC, `standard` (b) MC, `left` (c) MC, `speed` (d) MC, `height`



(e) RC, `speed` (f) RC, `clockwise` (g) RC, `c-clockwise` (h) RC, `radial`

Figure 11: Performance comparison in MC, RC, and HP for different tasks using **Action-based Policy Compression**.




(a) MC, `standard` (b) MC, `left` (c) MC, `speed` (d) MC, `height`

(e) RC, `speed` (f) RC, `clockwise` (g) RC, `c-clockwise` (h) RC, `radial`

(i) HP, `forward` (j) HP, `backward` (k) HP, `standstill` (l) HP, `standard`

Figure 12: Performance comparison in MC, RC, and HP for different tasks across all **baselines**.