K-MYRIAD: JUMP-STARTING REINFORCEMENT LEARNING WITH UNSUPERVISED PARALLEL AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Parallelization in Reinforcement Learning is typically used to speed up the training of a single policy, where multiple workers collect experience from the same sampling distribution. This common design limits the potential for parallelization by overlooking the benefits of diverse exploration strategies. We propose **K-Myriad**, a scalable and unsupervised method that maximizes the collective state entropy induced by a population of parallel policies. By cultivating a portfolio of specialized exploration strategies, K-Myriad provides a robust initialization for Reinforcement Learning, leading to both higher training efficiency and the discovery of heterogeneous solutions. Experiments on high-dimensional continuous control tasks with large-scale parallelization demonstrate that K-Myriad can learn a broad set of distinct policies, highlighting its effectiveness for collective exploration and paving the way for novel parallelization strategies.

1 INTRODUCTION

Despite its recent successes, most notably in fine-tuning large-scale language and reasoning models (Christiano et al., 2017; Guo et al., 2025), Reinforcement Learning (RL, Sutton & Barto, 2018) remains notoriously sample-inefficient. A key factor in unlocking its potential has been the advent of massive parallel computation, which speeds up experience collection by running independent copies of an environment in parallel. Training RL models on parallel processes is now standard. While this practice reduces wall-clock time by a factor proportional to the number of parallel processes, it does not fully exploit parallelization at the algorithmic level, which mostly follows the same design of a sequential computation setting (Espeholt et al., 2018; 2020).

Critically, little has been done to address the interplay between parallel computation and the root cause of RL's inefficiency, i.e., exploration. Recent works have provided some compelling ideas in this direction. Mayor et al. (2025) consider the trade-off between the length of each rollout and the number of parallel environments from which rollouts are sampled simultaneously. Their findings show that parallel environments yield more efficient use of computational resources than increasing the rollout length, even when the total number of collected samples is the same.

On an orthogonal line of work, De Paola et al. (2025) consider reward-free exploration in the parallel setting. While exploration is usually balanced with exploitation in the RL setting, in a reward-free one, there is no reward to exploit. In this scenario, a compelling exploration goal is to collect the most useful experience to solve any downstream RL task. A common and principled formulation of this goal is state-entropy maximization (Hazan et al., 2019), in which the agent aims to maximize the entropy of the distribution of state visitations induced by its policy. This has been shown to encourage the discovery of varied behaviors in an unsupervised manner, and to provide a useful initialization for RL (Mutti et al., 2021; Liu & Abbeel, 2021b; Seo et al., 2021; Yarats et al., 2021). However, most of these approaches consider a single generalist exploration policy. De Paola et al. (2025) propose learning a collection of policies that *jointly* maximizes the entropy of the induced state distribution while maintaining intra-ensemble specialization. Although they present both theoretical justification and encouraging empirical outcomes, their experimental validation is confined to toy environments with discrete state and action spaces and to a limited number of parallel processes.

In this paper, we address the problem of parallel state-entropy maximization in continuous domains and with massive parallelization. The main contribution of our work is the development of a practical and scalable algorithm, called **K-Myriad**, that can pretrain a large collection of policies (up to 50)

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

while interacting with hundreds or thousands of copies of the same environment. Furthermore, we show how policy collections can improve the efficiency of downstream RL across a variety of tasks. Compared with a single general-purpose maximum entropy policy, the collection of policies is more likely to include one well aligned with the task reward, regardless of the task. By directly evaluating the most promising policy in the collection for a given task and using it as the initial policy for a base RL algorithm, our approach outperforms the same base algorithm when initialized randomly or with a generalist maximum entropy policy. The **contributions** are organized as follows:

- We define the problem of state entropy maximization in large-scale parallel MDPs (Section 4);
- We propose a scalable policy gradient algorithm, K-Myriad, to address the introduced problem in challenging high-dimensional domains and over hundreds of parallel processes (Section 5);
- We propose a simple approach to jump-start RL with the trained parallel policies that adapts a common recipe of the non-parallel setting (Section 5.4);
- We present an experimental campaign on GPU-based parallel simulators of high-dimensional continuous control tasks, from locomotion to manipulation, developed in Isaac Sim (Section 6).

The complete codebase of our work is available here.

2 Related Works

State Entropy Maximization. Maximizing the entropy of state visitation in MDPs was first introduced by Hazan et al. (2019), inspiring a broad range of follow-up work on reward-free exploration (Lee et al., 2019; Mutti & Restelli, 2020; Mutti et al., 2021; 2022a;b; Mutti, 2023; Guo et al., 2021; Liu & Abbeel, 2021b;a; Seo et al., 2021; Yarats et al., 2021; Nedergaard & Cook, 2022; Yang & Spaan, 2023; Tiapkin et al., 2023; Jain et al., 2023; Kim et al., 2023; Zisselman et al., 2023; Zamboni et al., 2024a;b; 2025; De Paola et al., 2025; Ashlag et al., 2026). These works show the effectiveness of entropy maximization for policy pretraining (Liu & Abbeel, 2021b), as an intrinsic objective for exploration (Seo et al., 2021), and as a regularization to improve robustness (Ashlag et al., 2025).

Exploration with Multiple Agents. Concurrent exploration with multiple agents has been studied in settings where agents operate in the same environment (Guo & Brunskill, 2015; Parisotto et al., 2019; Jung et al., 2020; Chen et al., 2022; Qin et al., 2022). Other works investigate information sharing between agents (Alfredo & Arjun, 2017; Holen et al., 2023), or the theoretical foundations of coordinated exploration (Dimakopoulou & Van Roy, 2018; Dimakopoulou et al., 2018). In Multi-Agent Reinforcement Learning (MARL, Albrecht et al., 2024), exploration has often been driven by reward shaping and heuristic bonuses (Wang et al., 2019; Zhang et al., 2021; Xu et al., 2024). For instance, Zhang et al. (2023) propose an exploration bonus that maximizes deviation from jointly explored regions. Zamboni et al. (2025) explicitly address state entropy maximization in a multi-agent setting. A key distinction is that, in conventional MARL, agents act concurrently in a shared environment, and their trajectories are interdependent (Li et al., 2021; Zhao et al., 2021; Lupu et al., 2021). By contrast, our framework involves agents interacting with independent replicas of the environment. This structural difference eliminates inter-agent interference while allowing specialization to emerge, setting our work apart from the MARL literature.

Efficient Methods for Parallel Learning. A related line of work focuses on systems and architectures that scale RL via high-throughput parallel data collection and accelerator-aware training. IMPALA (Espeholt et al., 2018) decouples acting and learning by using a centralized learner and multiple distributed actors, employing V-trace for off-policy correction to enable stable learning at very high frame rates. SEED RL (Espeholt et al., 2020) further optimizes the pipeline through centralized, accelerator-backed inference and an efficient communication layer, attaining millions of frames per second at reduced cost. On single machines, Sample Factory (Petrenko et al., 2020) demonstrates aggressive CPU–GPU pipelining and asynchronous PPO variants, achieving 10^5+ FPS. In multi-agent settings, WarpDrive (Lan et al., 2022) executes both simulation and learning entirely on the GPU, eliminating CPU–GPU data transfers and scaling to thousands of concurrent environments and agents. These systems are complementary to our parallel MDP formulation: While our objective targets diversity and entropy maximization across parallel environment replicas, the above frameworks provide engineering patterns (centralized learners, accelerator-based inference, end-to-end GPU execution) that can be used to implement our algorithm efficiently at scale. In robotics, parallel simulators have also been shown to drastically shorten wall-clock time (e.g., Isaac-based setups,

Rudin et al., 2022; Mittal et al., 2023). Also related to our setting, but mostly orthogonal in spirit, the work by Mayor et al. (2025) analyzes the impact of parallel data collection in on-policy reinforcement learning, where the same policy is deployed across parallel environments.

Diverse Skill Discovery. The problem of learning a diverse set of skills is closely related to diversifying exploration through parallelization. Notable approaches include variational intrinsic control (Gregor et al., 2017), DIAYN (Eysenbach et al., 2019), and their extensions that maximize mutual information between skills and visited states (Hansen et al., 2019; Sharma et al., 2020; Campos et al., 2020; Liu & Abbeel, 2021a; He et al., 2022; Park et al., 2024). Zahavy et al. (2022) explicitly optimize for diversity among policies. These methods complement ours by introducing alternative heuristics to encourage behavioral diversity among agents, without relying on extrinsic rewards. Unlike ours, they do not consider the interplay between agents' diversity and parallel computation.

3 PRELIMINARIES

We denote $[N] := \{1, 2, \dots, N\}$. For a space $\mathcal{X}$, $\mathcal{X}^m$ is the product space of m copies of $\mathcal{X}$.

Entropy. For a measurable space $\mathcal{X}$, $\mathcal{P}(\mathcal{X})$ is the set of all probability density functions (PDFs) on $\mathcal{X}$. For a random variable X on $\mathcal{X}$ with PDF $f \in \mathcal{P}(\mathcal{X})$, its differential Shannon entropy is $H(X) = -\int_{\mathcal{X}} f(x) \log f(x)\, dx$. In continuous, high-dimensional settings, computing the latter integral can be non-trivial, as it requires characterizing the PDF f. Various methods exist to estimate the entropy $H(X)$ given realizations $\{x_i\}_{i=1}^N$ of the random variable X (Beirlant et al., 1997). A common approach (Singh et al., 2003) is to use a k-Nearest Neighbors (k-NN) estimator of the form

$$\hat{H}_k(f) = -\frac{1}{N} \sum_{i=1}^{N} \ln \frac{k}{N V_i^k} + \ln k - \Psi(k), \tag{1}$$

where Ψ is the digamma, $\ln k - \Psi(k)$ is a bias correction term, and V_i^k is the volume of the hypersphere of radius $R_i = \|x_i - x_i^{k\text{-NN}}\|$, which is the Euclidean distance between x_i and its k-NN $x_i^{k\text{-NN}}$. The k-NN estimator is known to be asymptotically unbiased and consistent (Singh et al., 2003).

MDP. The Markov Decision Process (MDP, Puterman, 2014) is a popular model for sequential decision making under uncertainty. The MDP is defined as $\mathbb{M} := (\mathcal{S}, \mathcal{A}, p, r, \mu, T)$ where $\mathcal{S} \subseteq \mathbb{R}^{d_s}$ is a continuous state space, $\mathcal{A} \subseteq \mathbb{R}^{d_a}$ is a continuous action space, p is a transition kernel such that $p(\cdot \mid s, a) \in \mathcal{P}(\mathcal{S})$ is the PDF of the next state given the current state and action pair $(s, a) \in \mathcal{S} \times \mathcal{A}$, r is a reward function such that $r(s, a) \in \mathbb{R}$ is the payoff for $(s, a) \in \mathcal{S} \times \mathcal{A}$, $\mu \in \mathcal{P}(\mathcal{S})$ is the initial state PDF, and $T \in \mathbb{N}^+$ is the finite horizon of an episode of interaction.

Policy and state distribution. An agent partially controls the MDP evolution through actions. A policy $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$ describes the strategy by which actions are selected, such that $\pi(a \mid s)$ is the conditional PDF of $a \in \mathcal{A}$ given $s \in \mathcal{S}$. Rolling out a policy over the MDP induces a distribution over trajectories $\tau = (s_0, a_0, \dots, s_{T-1}, a_{T-1}, s_T) \in \mathcal{K}$ with PDF $p_\pi(\tau) = \mu(s_0) \prod_{t=0}^{T-1} \pi(a_t \mid s_t) p(s_{t+1} \mid a_t, s_t)$, and a state visitation distribution $d_\pi \in \mathcal{P}(\mathcal{S})$ with density $d_\pi(s) = \mathbb{E}_{\tau \sim p_\pi}[\sum_{t=0}^{T-1} \mathbb{1}(s_t = s|\tau)/T]$, where $\delta(\cdot)$ denotes the Dirac delta function. Thus, $d_\pi(s)$ represents the density of visiting states in the continuous state space, marginalized over the steps of a trajectory of length T.

Objective. The typical objective of an MDP is to maximize the expected cumulative sum of rewards collected within a trajectory. Formally, $\max_{\pi \in \Pi} \mathbb{E}_{\tau \sim p^\pi}[\sum_{t=0}^{T-1} r(s_t, a_t)]$ where Π denotes the space of Markovian policies and π^* denotes the policy attaining the maximum, hence the *optimal policy*. The objective can be equivalently written as $\max_{\pi \in \Pi} \int_{\mathcal{S} \times \mathcal{A}} d_\pi(s) \pi(a|s) r(s, a) ds da$, which highlights that the problem is linear in the state distribution density d_π. RL is a family of methods to *learn* the optimal policy from sampled interactions with the MDP. RL methods typically employ the concept of value functions, which are $V_t^\pi(s) = \mathbb{E}_{\tau \sim p^\pi}[\sum_{j=t}^{T-1} r(s_j, a_j)|s_t = s]$ and $Q_t^\pi(s, a) = \mathbb{E}_{\tau \sim p^\pi}[\sum_{j=t}^{T-1} r(s_j, a_j)|s_t = s, a_t = a]$.

4 UNSUPERVISED EXPLORATION IN PARALLEL MDPS

We are interested in a setting where computational infrastructure enables the simulation of multiple copies of the same environment in parallel, which is the most common scenario for RL in practice. We formalize this setting as a Parallel Markov Decision Process (PMDP, Sucar, 2007), i.e., a collection

$$\mathbb{M}_p = (\mathbb{M}_i)_{i\in[m]} \qquad \text{where } \mathbb{M}_i = (\mathcal{S}, \mathcal{A}, p, \mu, T)$$

are $m \in \mathbb{N}^+$ identical and independent copies of the MDP $\mathbb{M}_i$.[1] The agent interacts with the PMDP by means of a *parallel policy*, which is a collection $\pi_p = (\pi_i)_{i\in[m]}$ of, possibly different, stochastic Markovian policies $\pi_i : \mathcal{S} \to \mathcal{P}(\mathcal{A})$, one for each copy of the MDP $\mathbb{M}_i$.

PMDP interaction protocol. At the start of the episode, the agent sends a policy π_i to the m parallel processes simulating the PMDP. For each process $i \in [m]$, an initial state is drawn $s_0^i \sim \mu$ independently. Then, for each step $t \in \{0, \dots, T-1\}$, an action $a_t^i \sim \pi_i(\cdot \mid s_t^i)$ is drawn from the respective policies independently, so that they trigger a transition in their copies of the MDP as $s_{t+1}^i \sim P(\cdot \mid s_t^i, a_t^i)$. Each parallel episode ends when the final state s_T^i is reached. The processes send the collected trajectories $\tau_i = (s_0^i, a_0^i, \dots, s_{T-1}^i, a_{T-1}^i, s_T^i)$ back to the agent, which gathers them into a *parallel trajectory* $\tau := (\tau_i)_{i\in[m]} \in \left(\mathcal{S}^{T+1} \times \mathcal{A}^T\right)^m$ stored with the previous experience, serving the basis for updating the policies to be sent to the processes in the subsequent episode.

4.1 THE PARALLEL STATE ENTROPY OBJECTIVE

The agent typically updates the parallel policy in the direction of the gradient of the RL objective, i.e., to maximize the expected return. Here, we consider a reward-free setting in which the agent can gather experience from the PMDP to pre-train a parallel policy before being tasked with a reward function across a variety of downstream tasks. To this end, following a recent problem formulation by De Paola et al. (2025), we aim to learn a collection of policies that is (a) *diverse*, to provide a robust initialization to downstream tasks and to implicitly drive RL to find alternative task solutions, (b) *exploratory*, such that the policies collectively cover the states of the environment, which allows us to identify rewarding regions and to learn from there.

Formally, De Paola et al. (2025) generalize the maximum state entropy objective (Hazan et al., 2019) to PMDPs. Let $d_{\pi_i} \in \mathcal{P}(\mathcal{S})$ the state distribution induced by the i-th component of the parallel policy π_p on the respective process i, i.e., $d_{\pi_i}(s) = \mathbb{E}_{\tau_i \sim p_{\pi_i}}[\sum_{t=0}^{T-1} \mathbb{1}(s_t = s|\tau_i)/T]$. Then, we formulate the *parallel* maximum state entropy objective as:

$$\max_{\pi_p \in \Pi^m} \mathcal{J}(\pi_p) = \max_{\pi_p \in \Pi^m} \left\{H(d_{\pi_p})\right\} \tag{2}$$

where $d_{\pi_p}(s) = \frac{1}{m}\sum_{i=1}^m d_{\pi_i}(s)$ is the *parallel state distribution*, the cumulative state distribution induced by the parallel policy π_p across the PMDP. How can (2) be optimized in practice by relying on sampled interactions with the environment? De Paola et al. (2025) proposed a flexible policy gradient scheme (Williams, 1992; Sutton et al., 1999; Peters & Schaal, 2008) that can be summarized in the following steps, repeated until convergence.

- **Collecting rollouts**: The parallel policy runs each component π_i in its respective environment copy to collect a parallel trajectory.
- **Entropy estimation**: The parallel trajectory is used to estimate the parallel state distribution d_{π_p} to then compute its entropy, or to estimate the entropy as $H(d_{\pi_p})$ directly.
- **Policy update**: Each component of π_p is updated in the direction of their respective gradient

$$\pi_i' \leftarrow \pi_i + \alpha \nabla_{\pi_i} H(d_{\pi_p}) \qquad \forall i \in [m]$$

De Paola et al. (2025) proposed this recipe, but they limited their analysis to toy domains with discrete states and actions and to at most a handful of parallel agents. It is less clear how to scale the recipe to more challenging domains, with high-dimensional continuous states and action spaces and massive parallelization, when hundreds or thousands of policies are deployed.

[1] For reasons that will be clear later, the MDP definition does not include the reward.

5 SCALABLE PARALLEL ALGORITHMS FOR THE STATE ENTROPY OBJECTIVE

Our main contribution is *K-Myriad*, a *scalable* algorithm for learning a population of policies that collectively maximize the state-entropy objective in (2) for continuous, high-dimensional domains in massively parallel environments. It follows the above policy-gradient scheme, with a scalability-focused implementation.

5.1 POLICY ARCHITECTURE

The first aspect to address when designing a scalable algorithm is how to represent the parallel policy. In principle, we would like to send a separate policy network to each parallel process. In practice, parallel processes are often run on the same GPU with shared memory, making it intractable to store dozens or hundreds of fully independent parameterizations. In K-Myriad, policies are implemented within a single policy network, composed of a shared trunk and m independent heads, one for each deployed policy (see Fig. 1). The shared trunk has two fully connected layers that process the state variables into a representation for the independent heads. The heads are shallow networks that take the representation and produce the mean and standard deviation of a Gaussian action distribution. This architectural design has two significant benefits: First, it allows for saving memory, as we need to store the parameters of the shared trunk once for all the processes; Second, it enables training a shared representation: The shared trunk collects gradients from all independent processes, while each parallel copy maintains its own action selection strategy. This single network architecture also samples actions across all parallel copies in a single forward pass, substantially speeding up computation. We empirically evaluate these learning speed gains in Apx. D.

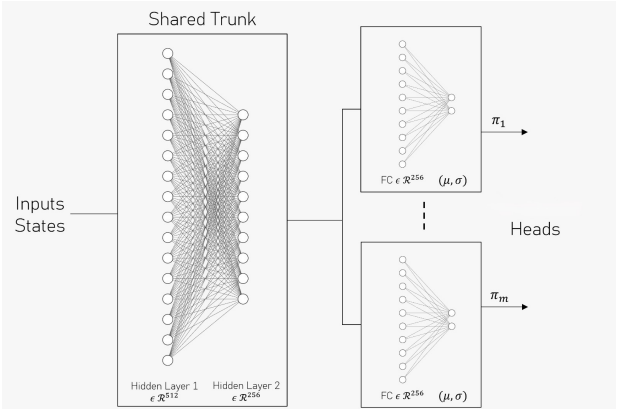


Figure 1: K-Myriad policy architecture. A shared backbone extracts features from the input, while independent heads produce distinct policy outputs.

5.2 LOSS FUNCTION

Having defined the policy architecture, we now need to specify a loss function to train the parameters of the shared trunk and the independent heads, thereby maximizing the state entropy objective (2). As it is common in prior works (e.g., Mutti et al., 2021; Liu & Abbeel, 2021b), we turn to a direct estimation of the entropy $H(d_{\pi_p})$ without an explicit estimate of the state distribution d_{π_p}, which is problematic in high-dimensional domains. We can use the kNN entropy estimator (1). For a parallel trajectory $\tau = ((s_0^i, a_0^i, \dots, s_{T-1}^i, a_{T-1}^i, s_T^i)_{i\in[m]})$, we estimate the entropy as follows

$$\hat{H}_k \propto \sum_{i=1}^{m} \sum_{t=0}^{T} \ln \|s_t^i - \text{kNN}(s_t^i)\|^d, \tag{3}$$

where $\text{kNN}(s_t^i)$ denotes the kNN of s_t^i.[2] Since the states are sampled by rolling out the parallel policy, they retain dependence on the policy parameters; in particular, the sampling of s_t^i depends on the parameters of head i and the shared trunk. We can differentiate Eq. 3 with respect to the policy parameters to perform a step in the gradient direction. We refer to Appendix B for how to differentiate Eq. 3. It is crucial to note that the $\text{kNN}(s_t^i)$ may be a state visited in a copy of the environment $j \neq i$, thereby incentivizing the independent heads to specialize in different regions of the state space. Here, the exponent d explicitly defines the state-space dimension, i.e., the cardinality of the set over which the entropy is measured.

[2]We omit all the constants, which are the same as in (1).

5.3 The Algorithm

We now have the elements to present K-Myriad, whose pseudocode is reported in Algorithm 1. First, a policy is initialized according to the architecture described in the previous sections. In one epoch of the main loop, each parallel process is assigned to a policy head. In the most straightforward case, the mapping is one-to-one, with a policy having m independent heads, each corresponding to a parallel process. In general, however, one can run the same head across multiple processes. This will be the case in our experiments, in which we will consider up to 1000 parallel processes for 10 or 50 heads. In this way, we can both reduce the total number of parameters and collect multiple trajectories from the same policy in a single simultaneous rollout. After the assignment, a rollout is collected in parallel for all processes, along with their respective policy heads. With the collected data, the loss is computed through the entropy estimation presented previously. Finally, the loss gradient is propagated in a single backward pass through the entire policy network, including the shared trunk and the independent heads. Once the epoch exceeds the budget E, the algorithm terminates and outputs the parallel policy.

Algorithm 1: `K-Myriad`

Input: Processes m, Policies $\overline{m}$, Rate α
Initialize the policy π_θ
for epoch $e = 1, \dots E$ **do**
  Policies to Processes:
  Assign a policy head to each process
  Data Collection:
  Collect a parallel rollout $\tau = (\tau_i)_{i \in [m]}$
  Entropy Estimation:
  Compute $\hat{H}_k$ as in Eq. 3 given τ
  Policy Gradient Update:
  $L(\theta) = \hat{H}_k$
  $\theta \leftarrow \theta + \alpha \nabla_\theta L(\theta)$
end for
Output: Parallel policy π_θ

5.4 Jump-starting RL

The output of Algorithm 1 is a set of parallel policies. It is not obvious how to use those policies to benefit learning downstream tasks in the same environment, which is the ultimate goal of unsupervised exploration. In single-policy settings, a simple but effective recipe is to initialize an RL algorithm with the pretrained policy (Mutti et al., 2021; Liu & Abbeel, 2021b). Following this scheme, which of the parallel policies should we use as the initial policy? We propose a simple strategy based on evaluating each pretrained policy using the task reward and then selecting the one with the highest initial performance. As we shall see in the experiments, this approach works surprisingly well in a variety of locomotion and manipulation tasks, as the diversity among the learned policies increases the likelihood of selecting an initial policy that is well aligned with the task. Nonetheless, more sophisticated strategies for exploiting parallel policies in downstream RL may be developed in future work.

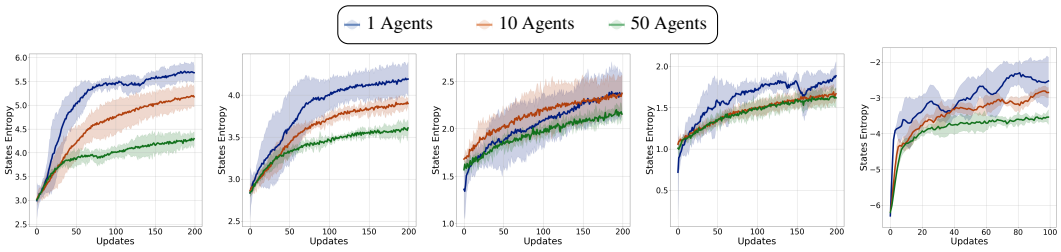
6 Experiments

To demonstrate the benefits of algorithm-level parallelization, we experiment with thousands of *Ant* and *Shadow Hand* (Melnik et al., 2021) replicas in Isaac Lab (Mittal et al., 2023), organized into:

- **Unsupervised Parallel Exploration**: We report the results of the learning process using Algorithm 1 and evaluate the performance of the learned policy as the number of agents grows.
- **Pretrained Locomotion and Manipulation**: We focus on RL for locomotion and manipulation problems, analyzing how pretraining parallel agents with heterogeneous policies, derived by optimizing (2) affects RL performance.

Details on the environments used in the experiments are provided in Apx. A. As a base environment for locomotion, we instantiate a robotic ant whose final objective is to learn to navigate in the *XY*-plane under various terrains and scenarios. In the Ant implementation of Isaac Sim, the robot is controlled by applying torques to the joints connecting the legs to the torso. The observation space consists of 62 features that capture the torso's position and velocity, as well as the contact forces experienced during locomotion. We design four environment variations of increasing difficulty: (i) an *Empty* terrain, (ii) a *Maze*, and terrains populated with *Caves* (iii) and *Pyramids* (iv).

To demonstrate the use of *K-Myriad* beyond locomotion, we consider a manipulation task in which a robotic hand holds a colored cube. The hand implementation in Isaac Sim has an observation space with 149 features, including the cube position and orientation, as well as features related to the finger position and velocity. The hand can manipulate the cube via 20 actuators applied at the finger level.



(a) Empty Env (b) Maze Env (c) Cave Env (d) Pyramid Env (e) Inhand Env

Figure 2: State entropy achieved by a single generalist agent and a collection of 10 or 50 agents. We report the average and 95% c.i. across multiple runs as a function of the policy updates.

6.1 Unsupervised Parallel Exploration

To evaluate the performance of the proposed method, we conduct experiments in Isaac Sim using $\tau = 1000$ parallel replicas of *Ant* or *Hands*. A varying number of agents $m \in \{1, 10, 50\}$ interact with those replicas as follows. At each epoch, one trajectory is drawn independently from each replica. When $m = 1$, all trajectories are generated using the same strategy. For $m > 1$, each agent draws a fraction τ/m of the trajectories. Once the interaction ends across all parallel replicas, all trajectories are aggregated, and the agents' policies are updated according to Alg. 1. Although we refer to *multiple agents* when $m > 1$, recall that their strategy is implemented through a single multi-head policy network rather than fully separate ones, as described in Section 5.

With this protocol, we compare the impact of different numbers of agents m across environments in terms of induced state entropy and per-agent diversity. For the sake of reproducibility, experiments are repeated across multiple seeds. Before commenting on the results below, note that we fix the total number of trajectories τ drawn per epoch to ensure fairness when comparing different numbers of agents. This means that the gradient of the single-agent policy is computed using τ trajectories, whereas the gradient of the parallel agents' policy heads is computed using τ/m trajectories each, inducing a trade-off between the gradient batch size and the parallel agents' diversity.

Figure 2 shows the state entropy achieved during the training epochs. First, we observe that the algorithm successfully optimizes the overall state entropy as the number of agents increases. Most importantly, **the groups of 10 Agents and 50 Agents achieve high state entropy with significant per-agent diversity**, as demonstrated in Table 1: For each agent, we collect a batch of 1000 trajectories and estimate the empirical KL divergence (Wang et al., 2009) between their state visitation distribution and those of all other agents. We then average the resulting divergences across agents. The diversity of the parallel agents can be further appreciated by visually inspecting the collected trajectories. Figure 3 shows a snapshot of the behavior produced by the 10 Agents: Parallel agents discover diverse behaviors in locomotion and manipulation. The Ant agents increase entropy by distributing their visits across distinct regions of the state space, whereas the hand agents increase entropy through diverse finger-movement strategies that yield different ways of rotating the cube.

The careful reader may have noted that 1 Agent version achieves the best performance across all domains in Figure 2, sometimes exceeding 10 Agents and 50 Agents. This is a product of the larger batch size per gradient update. Indeed, the populations of 10 agents and 50 agents also tend toward the top performance but at a slower convergence rate, especially in (a) Empty Env and (b) Maze Env, due to the reduced batch size per gradient update. The effect of the batch size is further analyzed in Apx. C.3, where we test 10 Agents with increasing batch size. The results indicate that 10 Agents and 50 Agents could match the performance of 1 Agent if given a larger batch size. Finally, we refer the reader to Appendix C for a detailed discussion of the k-NN implementation and extended analysis.

Table 1: Empirical KL divergence between parallel agents.

# agents	Empty	Maze	Pyramid	Cave	Inhand
10 Agents	0.99 ± 0.02	0.59 ± 0.03	0.53 ± 0.03	0.88 ± 0.04	1.79 ± 0.11
50 Agents	$\mathbf{1.33 \pm 0.01}$	$\mathbf{1.02 \pm 0.02}$	$\mathbf{0.88 \pm 0.03}$	$\mathbf{1.04 \pm 0.01}$	$\mathbf{2.69 \pm 0.05}$

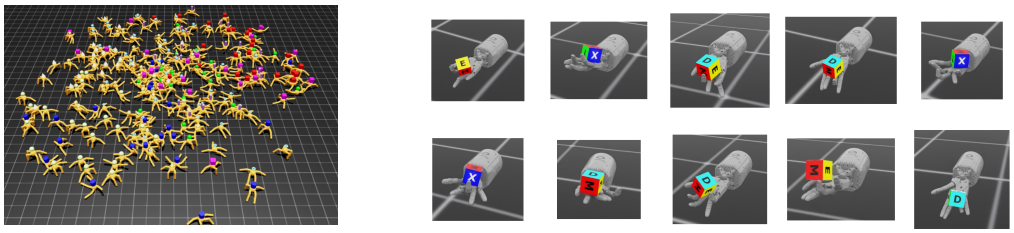


Figure 3: **Left:** State visitation distribution of Ant agents across parallel environments. **Right:** Diversity in cube orientations across *Inhand* task replicas controlled by 10 independent policy heads.

6.2 Pretrained Locomotion and Manipulation

To leverage the exploratory policies obtained with K-Myriad, we adopt a PPO (Schulman et al., 2017) fine-tuning scheme with the following structure. In all the previously presented environments, we compare three baselines: (i) a PPO agent with an actor policy that achieves the best success rate among 50 random parameter initializations, (ii) a PPO agent pretrained with a single-agent maximum-entropy policy, and (iii) a PPO agent initialized from the best among the 50 parallel heads pretrained as in the previous section. Each baseline policy is evaluated across 300 environment replicas for a total of 10M timesteps. The reward structure depends on the environment class: In **Locomotion**, agents receive a sparse reward of 1 upon entering a fixed radius around the goal. In **Manipulation**, success is defined by the agent's ability to orient a specific face of the cube upward.

Many strategies can be derived from the results identified by K-Myriad. To emphasize the useful degree of diversity achieved by each head for a single goal, we adopt a straightforward parallel policy extraction approach. When pretrained policies are available, we evaluate multiple rollout trajectories per head and select the head with the highest average success rate, converting it into a single-head actor. Otherwise, we evaluate multiple policies with randomly initialized parameters.

The results in Fig. 4 report success rates across goal locations (see the Appendix D for details). Overall, using the pretrained policies consistently outperforms RL with random initializations. Crucially, within the constrained interaction budget, a randomly initialized PPO agent must first learn basic motion primitives from scratch before it can begin learning to reach the goal.

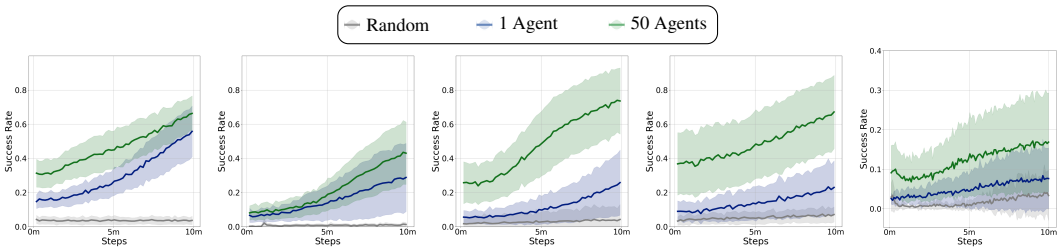
How does parallelization help in performance gain? **The key factor is diversity.**

The heads used to pretrain the PPO agent rapidly explore states close to the goal, accelerating convergence by generating rollouts enriched with highly effective, goal-oriented trajectories. In contrast, the single-agent baseline starts with a highly entropic policy that delays early learning, but eventually benefits from diverse rollouts to reach an acceptable success rate. Notably, as shown in Fig. 4, the absolute success rate for the agents on the Manipulation task is slightly lower than in the other environments. This is expected given the task's strict success criteria, which require precise cube orientation within a constrained timeframe. Nevertheless, the jump-started PPO algorithm demonstrates remarkable resilience and effectiveness, delivering superior initialization benefits and achieving a better final absolute mean success rate.

6.3 K-Myriad vs Unsupervised Skill discovery

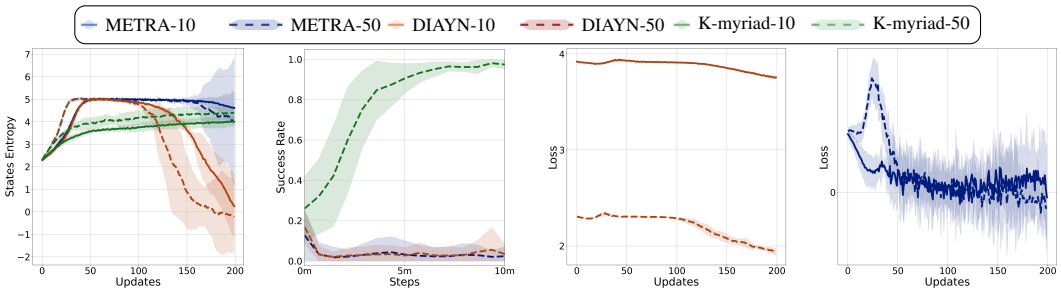
To position K-Myriad relative to existing SOTA methods, we conduct a dedicated analysis comparing it with DIAYN (Eysenbach et al., 2019) and METRA (Park et al., 2024). K-Myriad differs from these approaches in two main aspects. First, its policies are natively optimized in parallel rather than iteratively. Second, specialization emerges naturally from the shared state-coverage objective, without requiring an explicit skill-combination mechanism or additional hyperparameters to encourage diversity. The experiments are conducted in the *Ant Empty* locomotion environment described previously. The results are shown in Fig. 5, and the set of hyperparameters used is given in Apx. D.1.

The DIAYN loss (Fig. 5c) forces extreme skill separation that collapses the collective state entropy (Fig. 5a), resulting in inferior RL performance (Fig. 5b). Indeed, DIAYN maximizes the mutual information between skills and states. Through standard decomposition of the mutual information, one can see that the incentive to increase collective entropy, also maximized by K-Myriad, is counter-balanced by the one of minimizing the entropy of each individual skill in DIAYN. This is detrimental



(a) Empty Env (b) Maze Env (c) Cave Env (d) Pyramid Env (e) Inhand Env

Figure 4: PPO success rates, with different initialization strategies over multiple runs. Mean success rate over updates (solid line) with 95% confidence interval across runs (shaded area).



(a) State entropy (b) RL success rate (c) DIAYN loss (d) METRA loss

Figure 5: **Comparative evaluation of unsupervised skill discovery methods.** (a) State entropy in unsupervised exploration, (b) success rate in downstream RL tasks, (c) DIAYN discriminator loss $\mathcal{L}_{\text{disc}}$, and (d) METRA representation loss $\mathcal{L}_{\text{rep}}$. Solid curves show the mean across 4 seeds, shaded regions denote 95% confidence intervals.

when initializing an RL algorithm with those skills, which are too specialized and not a good starting point for learning. METRA avoids state entropy collapse via a temporally-aware metric, matching our high entropy findings (Fig. 5a). However, its policy learns high-momentum skills that fail to stabilize. In sparse goal-conditioned tasks, these behaviors induce sharp gradient updates and catastrophic forgetting at the goal, degrading downstream performance (Fig. 5b). K-Myriad maximizes collective state entropy without penalizing individual coverage and, in our analysis, provides a stronger pretraining for downstream RL than both DIAYN and METRA.

7 Conclusions

We introduced **K-Myriad**, a scalable unsupervised pretraining framework that uses a shared-trunk, multi-head architecture to maximize collective state entropy. Unlike conventional parallelization methods that optimize a single policy, **K-Myriad** promotes differentiation across agents, yielding a portfolio of complementary exploration strategies.

Limitations. Our method relies on large-scale parallel simulation, which may limit deployment in resource-constrained settings. Additionally, while increasing the number of agents might improve diversity, the currently proposed scheme might reduce per-agent sample efficiency as well, leading to slow convergence under a fixed interaction budget.

Future Directions. Our modular framework provides a scalable foundation for parallel RL pretraining, with clear paths toward improved policy reuse and broader domain generalization. Integrating differentiable parallel simulators could enable more efficient gradient-based learning and reduce the reliance on sample-intensive rollouts. Moreover, instead of collapsing pretrained policies into a single one, future work could fine-tune the full policy population jointly.

AI USE STATEMENT

In this work, we used generative AI tools exclusively for copy-editing, grammatical refinement, and sentence-level phrasing improvements. We have not used generative AI tools for literature synthesis, conceptual ideation, mathematical derivation, algorithmic formulation, experimental design. All AI-assisted edits were manually verified, reviewed, and finalized by the authors. We take full responsibility for the final content, claims, and artifacts presented in this work.

REPRODUCIBILITY STATEMENT

To ensure the reproducibility of our results, all algorithmic formulations, training protocols, and implementation specifics,including network architectures and the k-NN entropy estimator derivation,are detailed in the main text and Appendix. Complete hyperparameters, environment configurations, and per-seed results are systematically tabulated in the Appendix. Furthermore, fully documented and runnable source code along with scripts to reproduce all experiments are provided in the anonymous repository.

REFERENCES

Jiří Ajgl and Miroslav Šimandl. Differential entropy estimation by particles. *IFAC Proceedings Volumes*, 44(1):11991–11996, 2011. ISSN 1474-6670. doi: https://doi.org/10.3182/20110828-6-IT-1002.01404. URL https://www.sciencedirect.com/science/article/pii/S1474667016455436. 18th IFAC World Congress.

Stefano V. Albrecht, Filippos Christianos, and Lukas Schäfer. *Multi-Agent Reinforcement Learning: Foundations and modern approaches*. MIT Press, 2024.

C Alfredo and C Arjun. Efficient parallel methods for deep Reinforcement Learning. In *The Multi-disciplinary Conference on Reinforcement Learning and Decision Making (RLDM)*, pp. 1–6, 2017.

Yonatan Ashlag, Uri Koren, Mirco Mutti, Esther Derman, Pierre-Luc Bacon, and Shie Mannor. State entropy regularization for robust reinforcement learning. *arXiv preprint arXiv:2506.07085*, 2025.

Yonatan Ashlag, Mirco Mutti, Aviv Tamar, and Kfir Yehuda Levy. Probing in the dark: State entropy maximization for POMDPs. In *International Conference on Learning Representations*, 2026.

Jan Beirlant, Edward J. Dudewicz, László Györfi, and Istvan Denes. Nonparametric entropy estimation. an overview. 1997. URL https://api.semanticscholar.org/CorpusID:5722994.

Víctor Campos, Alexander Trott, Caiming Xiong, Richard Socher, Xavier Giró-i Nieto, and Jordi Torres. Explore, discover and learn: Unsupervised discovery of state-covering skills. In *International Conference on Machine Learning*, 2020.

Yan Chen, Perry Dong, Qinxun Bai, Maria Dimakopoulou, Wei Xu, and Zhengyuan Zhou. Society of agents: Regret bounds of concurrent thompson sampling. *Advances in Neural Information Processing Systems*, 35:7587–7598, 2022.

Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.

Vincenzo De Paola, Riccardo Zamboni, Mirco Mutti, and Marcello Restelli. Enhancing diversity in parallel agents: A maximum state entropy exploration story. *Proceedings of the International Conference on Machine Learning (ICML)*, 2025. URL https://arxiv.org/abs/2505.01336.

Maria Dimakopoulou and Benjamin Van Roy. Coordinated exploration in concurrent Reinforcement Learning. In *International Conference on Machine Learning*, pp. 1271–1279. PMLR, 2018.

Maria Dimakopoulou, Ian Osband, and Benjamin Van Roy. Scalable coordinated exploration in concurrent Reinforcement Learning. *Advances in Neural Information Processing Systems*, 31, 2018.

Lasse Espeholt, Hubert Soyer, Rémi Munos, Karen Simonyan, Volodymyr Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. *CoRR*, abs/1802.01561, 2018. URL http://arxiv.org/abs/1802.01561.

Lasse Espeholt, Raphaël Marinier, Piotr Stanczyk, Ke Wang, and Marcin Michalski. SEED RL: Scalable and efficient deep-RL with accelerated central inference. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rkgvXlrKwH.

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*, 2019.

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

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Zhaohan Guo and Emma Brunskill. Concurrent PAC RL. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29, 2015.

Zhaohan Daniel Guo, Mohammad Gheshlagi Azar, Alaa Saade, Shantanu Thakoor, Bilal Piot, Bernardo Avila Pires, Michal Valko, Thomas Mesnard, Tor Lattimore, and Rémi Munos. Geometric entropic exploration. *arXiv preprint arXiv:2101.02055*, 2021.

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.

Elad Hazan, Sham Kakade, Karan Singh, and Abby Van Soest. Provably efficient Maximum Entropy Exploration. In *International Conference on Machine Learning*, pp. 2681–2691. PMLR, 2019.

Shuncheng He, Yuhang Jiang, Hongchang Zhang, Jianzhun Shao, and Xiangyang Ji. Wasserstein unsupervised Reinforcement Learning. In *AAAI Conference on Artificial Intelligence*, 2022.

Martin Holen, Per-Arne Andersen, Kristian Muri Knausgård, and Morten Goodwin. Loss and reward weighing for increased learning in distributed Reinforcement Learning. *arXiv preprint arXiv:2304.12778*, 2023.

Arnav Kumar Jain, Lucas Lehnert, Irina Rish, and Glen Berseth. Maximum state entropy exploration using predecessor and successor representations. In *Advances in Neural Information Processing Systems*, 2023.

Whiyoung Jung, Giseung Park, and Youngchul Sung. Population-guided parallel policy search for Reinforcement Learning. *arXiv preprint arXiv:2001.02907*, 2020.

Dongyoung Kim, Jinwoo Shin, Pieter Abbeel, and Younggyo Seo. Accelerating Reinforcement Learning with value-conditional state entropy exploration. In *Advances in Neural Information Processing Systems*, 2023.

Tian Lan, Sunil Srinivasa, Huan Wang, and Stephan Zheng. Warpdrive: Fast end-to-end deep multi-agent reinforcement learning on a gpu. *Journal of Machine Learning Research*, 23:1–6, 2022. URL https://www.jmlr.org/papers/v23/22-0185.html.

Lisa Lee, Benjamin Eysenbach, Emilio Parisotto, Eric Xing, Sergey Levine, and Ruslan Salakhutdinov. Efficient exploration via state marginal matching. *arXiv preprint arXiv:1906.05274*, 2019.

Chenghao Li, Chengjie Wu, Tonghan Wang, Jun Yang, Qianchuan Zhao, and Chongjie Zhang. Celebrating diversity in shared multi-agent Reinforcement Learning. *CoRR*, abs/2106.02195, 2021. URL https://arxiv.org/abs/2106.02195.

Hao Liu and Pieter Abbeel. APS: Active pretraining with successor features. In *International Conference on Machine Learning*, 2021a.

Hao Liu and Pieter Abbeel. Behavior from the void: Unsupervised active pre-training. In *Advances in Neural Information Processing Systems*, 2021b.

Andrei Lupu, Brandon Cui, Hengyuan Hu, and Jakob Foerster. Trajectory diversity for zero-shot coordination. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 7204–7213. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/lupu21a.html.

Walter Mayor, Johan Obando-Ceron, Aaron Courville, and Pablo Samuel Castro. The impact of on-policy parallelized data collection on deep reinforcement learning networks. In *International Conference on Machine Learning*, 2025.

Andrew Melnik, Luca Lach, Matthias Plappert, Timo Korthals, Robert Haschke, and Helge Ritter. Using tactile sensing to improve the sample efficiency and performance of deep deterministic policy gradients for simulated in-hand manipulation tasks. *Frontiers in Robotics and AI*, 8:538773, 2021.

Mayank Mittal, Calvin Yu, Qinxi Yu, Jingzhou Liu, Nikita Rudin, David Hoeller, Jia Lin Yuan, Ritvik Singh, Yunrong Guo, Hammad Mazhar, Ajay Mandlekar, Buck Babich, Gavriel State, Marco Hutter, and Animesh Garg. Orbit: A unified simulation framework for interactive robot learning environments. *IEEE Robotics and Automation Letters*, 8(6):3740–3747, 2023. doi: 10.1109/LRA.2023.3270034.

Mirco Mutti. *Unsupervised Reinforcement Learning via state entropy maximization*. PhD Thesis, Università di Bologna, 2023.

Mirco Mutti and Marcello Restelli. An intrinsically-motivated approach for learning highly exploring and fast mixing policies. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 5232–5239, 2020.

Mirco Mutti, Lorenzo Pratissoli, and Marcello Restelli. Task-agnostic exploration via policy gradient of a non-parametric state entropy estimate. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 9028–9036, 2021.

Mirco Mutti, Riccardo De Santi, and Marcello Restelli. The importance of non-Markovianity in maximum state entropy exploration. In *International Conference on Machine Learning*, 2022a.

Mirco Mutti, Mattia Mancassola, and Marcello Restelli. Unsupervised Reinforcement Learning in multiple environments. In *AAAI Conference on Artificial Intelligence*, 2022b.

Alexander Nedergaard and Matthew Cook. k-Means Maximum Entropy Exploration. *arXiv preprint arXiv:2205.15623*, 2022.

Emilio Parisotto, Soham Ghosh, Sai Bhargav Yalamanchi, Varsha Chinnaobireddy, Yuhuai Wu, and Ruslan Salakhutdinov. Concurrent meta Reinforcement Learning. *arXiv preprint arXiv:1903.02710*, 2019.

Seohong Park, Oleh Rybkin, and Sergey Levine. Metra: Scalable unsupervised rl with metric-aware abstraction. In *International Conference on Learning Representations*, 2024.

Jan Peters and Stefan Schaal. Reinforcement Learning of motor skills with policy gradients. *Neural Networks*, 2008.

Aleksei Petrenko, Zhehui Huang, Tushar Kumar, Gaurav Sukhatme, and Vladlen Koltun. Sample factory: Egocentric 3d control from pixels at 100000 fps with asynchronous reinforcement learning. In *Proceedings of the 37th International Conference on Machine Learning*, pp. 7652–7662. PMLR, 2020. URL https://proceedings.mlr.press/v119/petrenko20a.html.

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

Chao Qin, Zheng Wen, Xiuyuan Lu, and Benjamin Van Roy. An analysis of ensemble sampling. *Advances in Neural Information Processing Systems*, 35:21602–21614, 2022.

Nikita Rudin, David Hoeller, Philipp Reist, and Marco Hutter. Learning to walk in minutes using massively parallel deep reinforcement learning. In *Conference on Robot Learning*, pp. 91–100. PMLR, 2022.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

Younggyo Seo, Lili Chen, Jinwoo Shin, Honglak Lee, Pieter Abbeel, and Kimin Lee. State entropy maximization with random encoders for efficient exploration. In *International Conference on Machine Learning*, 2021.

Archit Sharma, Shixiang Gu, Sergey Levine, Vikash Kumar, and Karol Hausman. Dynamics-aware unsupervised discovery of skills. In *International Conference on Learning Representations*, 2020.

Harshinder Singh, Neeraj Misra, Vladimir Hnizdo, Adam Fedorowicz, and Eugene Demchuk. Nearest neighbor estimates of entropy. *American Journal of Mathematical and Management Sciences*, 23 (3-4):301–321, 2003. doi: 10.1080/01966324.2003.10737616. URL https://doi.org/10.1080/01966324.2003.10737616.

L Enrique Sucar. Parallel markov decision processes. *Advances in Probabilistic Graphical Models*, pp. 295–309, 2007.

Richard S Sutton and Andrew G Barto. *Reinforcement Learning: An introduction*. MIT press, 2018.

Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for Reinforcement Learning with function approximation. In *Advances in Neural Information Processing Systems*, 1999.

Daniil Tiapkin, Denis Belomestny, Daniele Calandriello, Eric Moulines, Remi Munos, Alexey Naumov, Pierre Perrault, Yunhao Tang, Michal Valko, and Pierre Menard. Fast rates for maximum entropy exploration. In *International Conference on Machine Learning*, pp. 34161–34221. PMLR, 2023.

Qing Wang, Sanjeev R Kulkarni, and Sergio Verdú. Divergence estimation for multidimensional densities via k-nearest-neighbor distances. *IEEE Transactions on Information Theory*, 55(5): 2392–2405, 2009.

Tonghan Wang, Jianhao Wang, Yi Wu, and Chongjie Zhang. Influence-based multi-agent exploration. *arXiv preprint arXiv:1910.05512*, 2019.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist Reinforcement Learning. *Machine Learning*, 8:229–256, 1992.

Pei Xu, Junge Zhang, and Kaiqi Huang. Population-based diverse exploration for sparse-reward multi-agent tasks. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, pp. 283–291, 2024.

Qisong Yang and Matthijs TJ Spaan. CEM: Constrained entropy maximization for task-agnostic safe exploration. In *AAAI Conference on Artificial Intelligence*, 2023.

Denis Yarats, Rob Fergus, Alessandro Lazaric, and Lerrel Pinto. Reinforcement Learning with prototypical representations. In *International Conference on Machine Learning*, 2021.

Tom Zahavy, Yannick Schroecker, Feryal Behbahani, Kate Baumli, Sebastian Flennerhag, Shaobo Hou, and Satinder Singh. Discovering policies with DOMiNO: Diversity optimization maintaining near optimality. *arXiv preprint arXiv:2205.13521*, 2022.

Riccardo Zamboni, Duilio Cirino, Marcello Restelli, and Mirco Mutti. How to explore with belief: state entropy maximization in POMDPs. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024a.

Riccardo Zamboni, Duilio Cirino, Marcello Restelli, and Mirco Mutti. The limits of pure exploration in POMDPs: When the observation entropy is enough. *RLJ*, 2:676–692, 2024b. URL https://rlj.cs.umass.edu/2024/papers/Paper95.html.

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

Shaowei Zhang, Jiahan Cao, Lei Yuan, Yang Yu, and De-Chuan Zhan. Self-motivated multi-agent exploration. *arXiv preprint arXiv:2301.02083*, 2023.

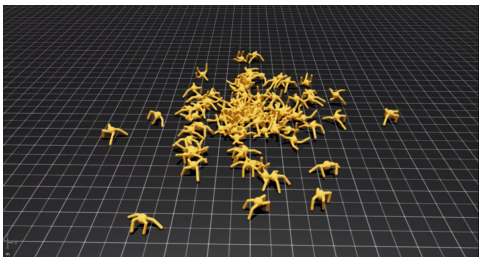
Tianjun Zhang, Paria Rashidinejad, Jiantao Jiao, Yuandong Tian, Joseph E Gonzalez, and Stuart Russell. MADE: Exploration via maximizing deviation from explored regions. *Advances in Neural Information Processing Systems*, 34:9663–9680, 2021.

Rui Zhao, Jinming Song, Haifeng Hu, Yang Gao, Yi Wu, Zhongqian Sun, and Yang Wei. Maximum entropy population based training for zero-shot human-ai coordination. *CoRR*, abs/2112.11701, 2021. URL https://arxiv.org/abs/2112.11701.

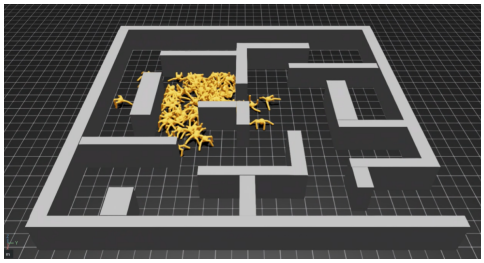
Ev Zisselman, Itai Lavie, Daniel Soudry, and Aviv Tamar. Explore to generalize in zero-shot RL. In *Advances in Neural Information Processing Systems*, 2023.

A Environment Configurations

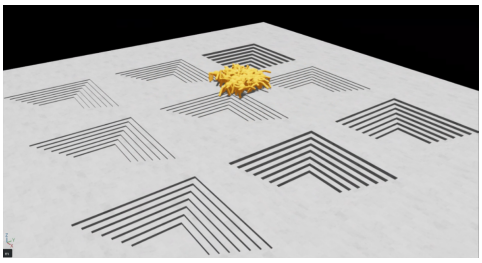
We design our experiments on top of `IsaacLab` (Mittal et al., 2023), using a manager-based environment with modular configuration classes. The environments differ primarily in the terrain setup, while sharing the same agent embodiment (the quadruped Ant robot) and task specification.



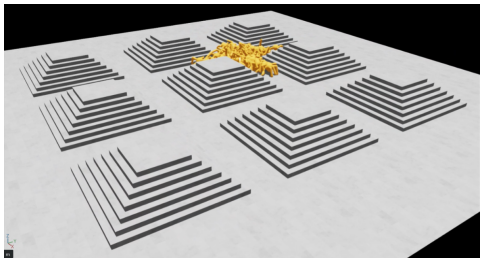
(a) Scene with an Empty Environment



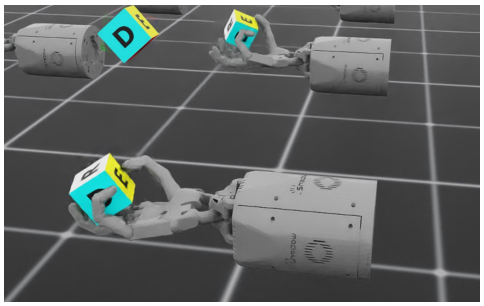
(b) Scene with a Maze Environment



(c) Scene full of Caves



(d) Scene full of Pyramids



(e) Shadow hand rolling cubes

Figure 6: Overview of the environments used: (a) flat ground with an Ant robot, (b) complex maze environment, (c) collection of pyramids, (d) cave-like terrain, and (e) robotic hand playing with a cube.

A.1 Robots and Scene

Ant scene The agent is a simulated Ant robot placed in an interactive scene. Each environment spawns a terrain importer, with optional additional assets (e.g., a maze). The terrain determines the complexity of locomotion and exploration:

- Empty Environment. A flat plane is used as the terrain. The surface has standard physical properties, making it suitable for baseline locomotion experiments.
- Maze Environment. The terrain remains a flat plane, but a static maze structure is imported. The Ant must navigate corridors and obstacles, emphasizing exploration and spatial awareness.
- Cave Environment. The terrain is procedurally generated with irregular cave structures. This environment increases locomotion difficulty by requiring robust control to overcome slopes and low surfaces.

- Pyramid Environment. The terrain is procedurally generated with irregular elevations and pyramid-like structures. This environment increases locomotor difficulty by requiring robust control to overcome slopes and uneven terrain.

InHand scene The in-hand environment simulates a ShadowHand dexterous robot manipulating a DexCube object. The hand is mounted upright (palm facing up) with gravity enabled on the cube, creating the challenge of keeping the object in-hand while reorienting it.

A.2 Simulation Parameters

Simulations run at a physics step of $\Delta t = 1/60$ seconds, with a control frequency given by a decimation factor of 5. The physics engine uses PhysX settings with a bounce threshold velocity of 0.2, static/dynamic friction set to 1.0, and no restitution. All environments are highly parallelized, with up to 1000 simulated instances in a single rollout.

A.3 Action Space

The agent controls all actuated joints of the Ant using an *effort-based action space*. The action vector $a \in [-1, 1]^8$ is scaled by a factor of 7.5 to generate joint torques. This low-level continuous action space ensures direct control over locomotion dynamics.

A.4 Observations

Ant-like environments The policy receives a concatenation of state features, including:

- Root position and height of the base.
- Linear and angular velocities.
- Yaw and roll orientation.
- Relative angle and heading projection with respect to a distant target.
- Normalized joint positions and relative joint velocities.
- Contact forces on all four feet.
- Previous actions.

These observations provide both proprioceptive and exteroceptive signals, allowing the policy to reason about stability, locomotion, and task progression. The shape of the observations thus results in $[1000, 62]$, the number of parallel replica environments with the corresponding ant observation size. Although the relative angle and heading projection with respect to a fixed distant target are not required for the maximum-entropy objective, we retain the default task configuration from the isaac sim repository to ensure reproducibility.

Manipulation The observation vector received to actuate the hand is :

- Object root position relative to the environment origin (x, y, z)
- Object orientation as a quaternion (w, x, y, z)
- Object linear velocity in world frame
- Object angular velocity in world frame (scaled by 0.2)
- All robot joint positions normalized to their soft limits (range [-1, 1])
- All robot joint velocities relative to default velocities (scaled by 0.2)
- Positions of the 5 fingertip bodies in the environment frame, flattened
- Orientations (quaternions) of the 5 fingertip bodies, flattened
- Linear + angular velocities (6D) of the 5 fingertip bodies, flattened
- The previous action applied to the robot (20 actuated joint targets)

A.5 REWARDS

The reward function is used exclusively in the experiments evaluating the exploratory policy for downstream tasks.

Ant rewards Specifically, a sparse reward of $+1$ is assigned to the agent when it reaches the target position, defined as being within a Euclidean distance of less than $1\,\mathrm{m}$ from the goal in the environment described above.

Hand rewards We define a set of goals corresponding to the target quaternion orientations of the cube faces, excluding the initialization orientation. The agent receives a sparse reward of $+1$ if the current orientation is within the threshold of $\pm 5^\circ$ from the target goal (Fig.12).

A.6 TERMINATION CONDITIONS

The termination conditions vary depending on the task:

- *Ants* - episodes end only when the maximum time horizon T is reached, both in the reward free and goal based experiments.
- *Inhand* - the episode terminates either on timeout, if the cube falls from its default in-hand position, or upon successful alignment in the goal based setting.

B POLICY ARCHITECTURE

The policy architecture follows a *shared trunk with agent-specific heads* design. A single feed-forward backbone network processes the input state into a shared latent embedding, which is then specialized for each agent via lightweight adapters and multi-head output layers. This design enables parameter sharing across agents while still allowing for agent-specific action distributions.

Shared Trunk. The shared trunk is a multilayer perceptron (MLP) with ReLU activations:

$$x = f_\theta(s), \quad f_\theta : \mathbb{R}^{d_s} \to \mathbb{R}^{d_h},$$

where s is the input state, d_s the state dimension, and d_h the hidden size of the last trunk layer. This embedding x is common to all agents.

Agent-Specific Adapters. For each agent $i \in \{1, \ldots, m\}$, an adapter network

$$z_i = g_i(x), \quad g_i : \mathbb{R}^{d_h} \to \mathbb{R}^{d_a}$$

projects the shared embedding into an agent-specific representation. This lightweight specialization layer allows agents to adapt the shared features to their own behavioral characteristics.

Multi-Head Outputs. The final action distribution is produced by vectorized multi-head linear layers. For each agent i, we compute:

$$\mu_i = W_i^{(\mu)} z_i + b_i^{(\mu)}, \qquad \log \sigma_i = W_i^{(\sigma)} z_i + b_i^{(\sigma)},$$

where $W_i^{(\mu)}, W_i^{(\sigma)}$ and biases are head-specific parameters. The outputs (μ_i, σ_i) parameterize a Gaussian policy $\pi_i(a|s) = \mathcal{N}(\mu_i, \sigma_i^2)$, followed by a $\tanh$-squashing and affine rescaling to match the action space.

Importance of Sharing. By sharing the trunk f_θ across all agents, the network exploits common structure in the dynamics and observations, while the per-agent heads preserve diversity.

Loss Calculation We now describe how we compute gradients of the maximum-entropy objective in Eq. 1 with respect to the parameters of the parallel policies. Directly differentiating entropy over high-dimensional state distributions is intractable; therefore, following prior work, we adopt a non-parametric k-NN entropy estimator augmented with *importance weighting* Ajgl & Šimandl (2011). This formulation enables principled gradient computation with respect to policy parameters while remaining sample efficient.

The resulting entropy estimator is given by:

$$\hat{H}_k(d_{\pi_p}) = -\sum_{j=1}^{N} \left[\frac{W_j}{k} \log \frac{W_j}{V_j} \right] + \log k - \Psi(k), \tag{4}$$

where W_j denotes the aggregated importance weight of the k nearest neighbors of sample s_j, and V_j is the volume of the corresponding neighborhood induced by the k-NN radius.

The gradient of the objective with respect to θ is:

$$\nabla_\theta J(\theta) = \nabla_\theta \mathbb{E}_{d_{\pi_p} \sim \beta} \left[\hat{H}_k(d_{\pi_p}) \right], \tag{5}$$

where β is the behavioral policy used to collect trajectories. By applying the log-derivative trick, we obtain:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{j=1}^{N} \nabla_\theta W_j(\theta) \cdot \frac{\partial \hat{H}_k}{\partial W_j}. \tag{6}$$

The per-trajectory importance weight is computed as:

$$w_t^{(j)} = \exp\left(\sum_{u=0}^{t} \log \pi_i(a_u \mid s_u) - \log \beta_i(a_u \mid s_u) \right). \tag{7}$$

In practice, β_i is defined as a snapshot of π_i at the beginning of each rollout phase, while retaining the online nature of the algorithm. As a result, all importance weights evaluate to one, making the estimator numerically equivalent to the standard (unweighted) k-NN entropy estimator. The parameter update is weighted by the k-NN distances and the neighborhood volumes V_j, which are treated as constants (detached from the computational graph) during differentiation with respect to the weights W_j.

C K-NN Calculation and Scalability Details

The implementation of K-Myriad requires efficient k-Nearest Neighbor (k-NN) searches to compute the parallel state entropy and its gradients. In high-dimensional continuous domains with massive parallelization (up to 1,000 environment replicas), the number of state particles N can grow significantly, making a naive $O(N^2)$ distance matrix computation computationally prohibitive.

C.1 Distributed Search Strategy

To maintain scalability, the algorithm employs a unified reshaping method to compute k-NN distances globally across all trajectories and environments. This ensures that the local density around a state s_t^i is influenced by visits from other specialized policy heads, driving the required specialization across the population.

- **Computational Framework:** The search is performed using a combination of GPU-accelerated operations and highly optimized CPU-based spatial data structures. For $N \leq 60,000$, a batched pairwise Euclidean distance matrix is computed directly on the GPU using `torch.cdist` to leverage tensor cores. For larger datasets, the search utilizes the `cKDTree` implementation from SciPy or `NearestNeighbors` from Scikit-learn to maintain $O(N \log N)$ average-case complexity.
- **Memory Management:** To avoid Out-of-Memory (OOM) errors during the policy update, the k-NN distances and importance weights are processed in chunks, typically between 1,024 and 8,192 rows. This chunked approach allows the algorithm to scale to millions of timesteps without exceeding typical GPU VRAM limits.

C.2 ENTROPY VS k HYPERPARAMETER

In this section, we analyze the robustness of the k-nearest neighbor (k-NN) entropy estimator with respect to the hyperparameter k. Although the estimator is theoretically asymptotically unbiased and consistent, the choice of k induces a bias–variance trade-off. Smaller values of k yield highly local density estimates that are more sensitive to sampling noise, whereas larger values produce smoother estimates by aggregating information over broader regions of the state space. Figure 7 illustrates how the estimated collective state entropy varies as k changes across the *Empty*-environment.

Understanding this dependency is important to ensure that the intrinsic reward signal remains stable during the massively parallel training regime of K-MYRIAD. We evaluate a multi-head policy trained with 10 parallel agents under different values of k and observe that increasing k does not yield meaningful gains in entropy growth over training. In contrast, smaller k values provide comparable performance while keeping computational overhead low, making them more suitable for our experimental setting.

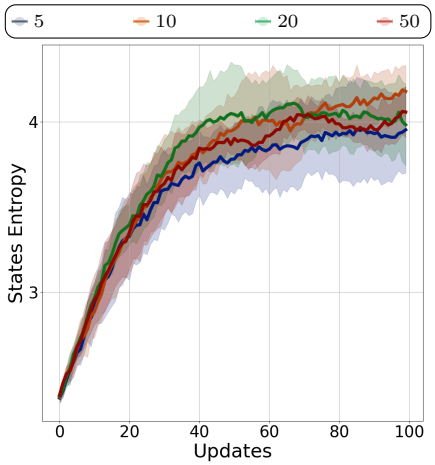


Figure 7: Parallel maximum entropy objective with 10 parallel agents, with different k-NN values. Mean State Entropy (solid line) and 95 % c.i. (shaded area)

C.3 TRAJECTORY BUDGET

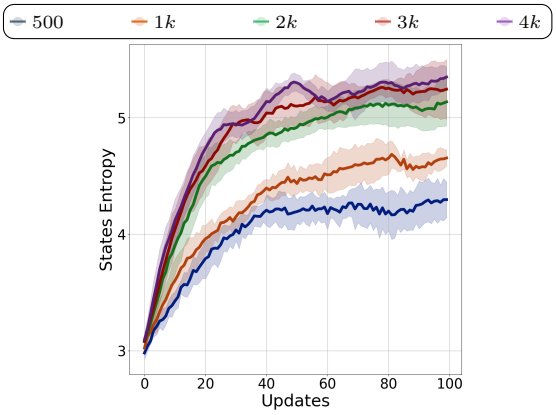


Figure 8: Trajectory-budget scaling experiment in the *Empty* environment.

Table 2: `Reward-Free` K-myriad parameters

Parameter	**Symbol**	`Single Agent`	`10 Agents`	`50 Agents`
Number of agents	a	1	10	50
Total trajectories (overall)	N_{traj}	1000	1000	1000
Number of environments replica	N_{env}	1000	1000	1000
Trajectories per agent	N_{traj}/a	1000	100	20
Trajectory length	T	600	600	600
Seeds	$\mathcal{S}$	$\{0, 1, 56, 123\}$	$\{0, 1, 56, 123\}$	$\{0, 1, 56, 123\}$
Policy architecture (Trunk hidden sizes)	–	[512, 256]	[512, 256]	[512, 256]
Policy architecture (Head adapters)	–	$1 \times [256, 256, 8]$	$10 \times [256, 256, 8]$	$50 \times [256, 256, 8]$
Learning rate	α	2×10^{-4}	2×10^{-4}	5×10^{-4}
LR scheduler (milestones, γ)	–	[30, 80], 0.5	[30, 80], 0.5	[30, 80], 0.5
k-NN (KNN step)	k	5	5	5

D Additional Experiments Details

Reward free hyperparameters Table 2 summarizes the task and algorithm configurations used for training in the single-, 10-, and 50-agent experiments described in Section 6. The single-agent baseline uses the same random seed, total number of trajectories, and trajectory length as the multi-agent setups, but differs in the number of trajectories processed per update and in the policy head structure.

State Visitation Heatmaps Figure 13 reports the collection of state-visitation heatmaps obtained at the final training iteration for the different pretraining strategies described in Section 6. These visualizations correspond closely to the entropy trends shown in Figure 2, providing spatial insight into the most frequently visited regions across environments. In the simpler *Empty* and *Maze* environments, the single-agent and 10-agent versions of K-Myriad exhibit broader exploration and higher performance compared to the 50-agent implementation. However, this ability to explore diverse states is significantly reduced in more complex settings such as *Pyramid* and *Cave*. In these environments, successful behavior requires not only expanding coverage in the X–Y plane but also along the Z-axis, such as climbing the pyramid or descending into the cave, resulting in higher-performing agents that achieve a wider spatial perimeter of visited states.

Goal based experiments hyperparameters Table 3 lists the PPO hyperparameters and corresponding configurations used in Section 6 for comparing the performance of the different pretraining strategies. To mitigate catastrophic forgetting during downstream tuning, we adjust the entropy coefficient (β_{ent} from 0.001 to 0.0) and clipping range (ϵ from 0.20 to 0.15) for the pretrained configurations. Additionally, the PPO epochs per rollout (N_{epoch}) are set to 5 and warmup rollouts to 5 for the pretrained cases, compared to 10 epochs and 1 warmup rollout in the non-pretrained baseline, to account for the accelerated convergence of pretrained agents.While the baseline and pretrained configurations use different PPO epoch settings (10 epochs vs. 5 epochs), this adaptation is intentional. Because the pretrained policy already contains informative examples from the pretraining phase, it exhibits higher sample efficiency and faster convergence. Consequently, setting $N_{\text{epoch}} = 10$ for the pretrained case leads to overfitting, while $N_{\text{epoch}} = 5$ is sufficient to stabilize the policy during downstream tuning without catastrophic forgetting.

Computational Resources All experiments were conducted on a local workstation equipped with a 12th Gen Intel(R) Core(TM) i9-12900E CPU, 64 GiB of RAM, and an NVIDIA RTX 5000 GPU (24GiB). To speed up experiments, we also work on cloud workstations provisioned through Brev, equipped with a single NVIDIA A6000 GPU (48 GiB), 28 CPU cores, and 58 GiB of system memory, running on Hyperstack infrastructure. Together, these environments provided sufficient GPU memory and parallel compute capabilities to train and evaluate reinforcement learning agents at scale. With the introduced computational architecture, we report the wall clock time for the experiments provided:

Table 3: PPO parameters

Parameter	Symbol	No Pretrain	1-Agent Pretrain	50-Agents Pretrain
Policy hidden sizes	H_π	[512, 256, 256]	[512, 256, 256]	[512, 256, 256]
Value function hidden sizes	H_V	[256, 256]	[256, 256]	[256, 256]
Policy learning rate	LR_π	1×10^{-5}	1×10^{-5}	1×10^{-5}
Value function learning rate	LR_V	3×10^{-4}	3×10^{-4}	3×10^{-4}
Discount factor	γ	0.99	0.99	0.99
GAE parameter	λ	0.95	0.95	0.95
Clipping range	ϵ	0.20	0.15	0.15
Entropy coefficient	β_{ent}	0.001	0.0	0.0
Value loss coefficient	β_V	0.5	0.5	0.5
Max gradient norm	$\|\nabla\|_{\max}$	0.5	0.5	0.5
PPO epochs per rollout	N_{epoch}	10	5	5
Warmup Rollout		1	5	5
Minibatch size	$B_{\min}$	$64 \times num_envs$	$64 \times num_envs$	$64 \times num_envs$
Rollout length	T	600	600	600
Number of environments replica	N_{env}	300	300	300
Total timesteps	N_{steps}	1×10^7	1×10^7	1×10^7

- Maximum Entropy Empty Environment: 200 epochs, 1000 replicas, 120M Timesteps, [1,10,50] agents approximately *40min* per run.
- Maximum Entropy Maze Environment: 200 epochs, 1000 replicas, 120M Timesteps, [1,10,50] agents approximately *51min* per run.
- Maximum Entropy Pyramid Environment: 200 epochs, 1000 replicas, 120M Timesteps,[1,10,50] agents approximately *48min* per run.
- Maximum Entropy Cave Environment: 200 epochs, 1000 replicas, 120M Timesteps, [1,10,50] agents approximately *44min* per run.
- Maximum Entropy Empty Environment: 1000 epochs, 1000 replicas, 600M Timesteps, [1,10,50] agents approximately *3h* per run.
- Maixmum Entropy Inhand Environment: 100 epochs, 1000 replicas, 300M Timesteps, [1,10,50] agents approximately *28min*
- Goal Based PPO training Empty Environment: 300 replicas, 10M Timesteps, [no-pretrain,1,50] agents approximately *25min* per run.
- Goal Based PPO training Inhand Environment: 300 replicas, 10M Timesteps, [no-pretrain,1,50] agents approximately *30min* per run.
- Goal Based PPO training Empty Environment: 300 replicas, 15M Timesteps, [no-pretrain] agents approximately *40min* per run.

Computation Time and Efficiency The central goal of our work is to make large-scale, reward-free pretraining feasible on modest, single-GPU hardware with limited computational budget. To this end, we evaluate the wall-clock performance of our multi-head network. Using a shared trunk yields a $12\times$ speed-up in inference time (i.e., sampling actions across all parallel networks) and an $8.4\times$ speed-up in the loss computation and gradient update for each epoch. Table 5 reports the average computation time per environment step and policy update.

D.1 DIAYN AND METRA HYPERPARAMETERS

To guarantee an unbiased comparison, both DIAYN (Eysenbach et al., 2019) and METRA (Park et al., 2024) are trained using the exact same multihead architecture and environment budget as our method. Policies share a $[512, 256]$ MLP trunk with a parallel $[256, 256]$ head adapters, collecting 1,000 parallel trajectories of horizon $T = 300$ per epoch across 200 epochs (6×10^7 environment steps in total).

Table 4: Pretraining hyperparameters for DIAYN and METRA multihead skill discovery.

Parameter	**Symbol**	10 Agents	50 Agents
Environment & Rollout			
Number of skills / agent heads	a	10	50
Number of parallel environment replicas	N_{env}	1000	1000
Total trajectories per epoch	N_{traj}	1000	1000
Parallel rollouts per skill	N_{traj}/a	100	20
Trajectory horizon	T	300	300
Pretraining epochs	E	200	200
Random seeds	$\mathcal{S}$	$\{0, 1, 56, 123\}$	
State filtering subspace (coordinates)	–	$[x, y]$ $(d = 2)$	
Policy Architecture			
Trunk network hidden layers	–	$[512, 256]$ (ReLU)	
Action output heads (mean & $\log \sigma$)	–	$10 \times [256, 256, 8]$	$50 \times [256, 256, 8]$
Optimization & Algorithm			
Optimizer	–	Adam	
Policy learning rate (trunk & heads)	α_π	3×10^{-4}	3×10^{-4}
Auxiliary learning rate (Disc. / Rep.)	α_{aux}	1×10^{-4}	1×10^{-4}
Discount factor	γ	0.99	0.99
Action entropy regularizer coefficient	β_{ent}	0.01	0.01
Method-Specific Architectures & Loss			
DIAYN Discriminator architecture $q(z \mid s)$	–	[2, 256, 256, 10]	[2, 256, 256, 50]
METRA Representation architecture $\phi(s)$	–	[2, 256, 256, 2]	
METRA Lipschitz penalty weight	λ	1.0	
Evaluation Metric			
k-NN entropy parameter	k	5	5

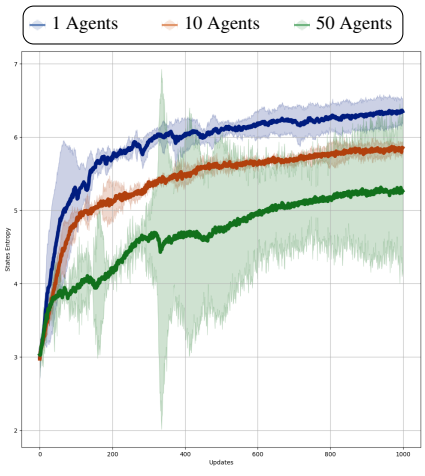
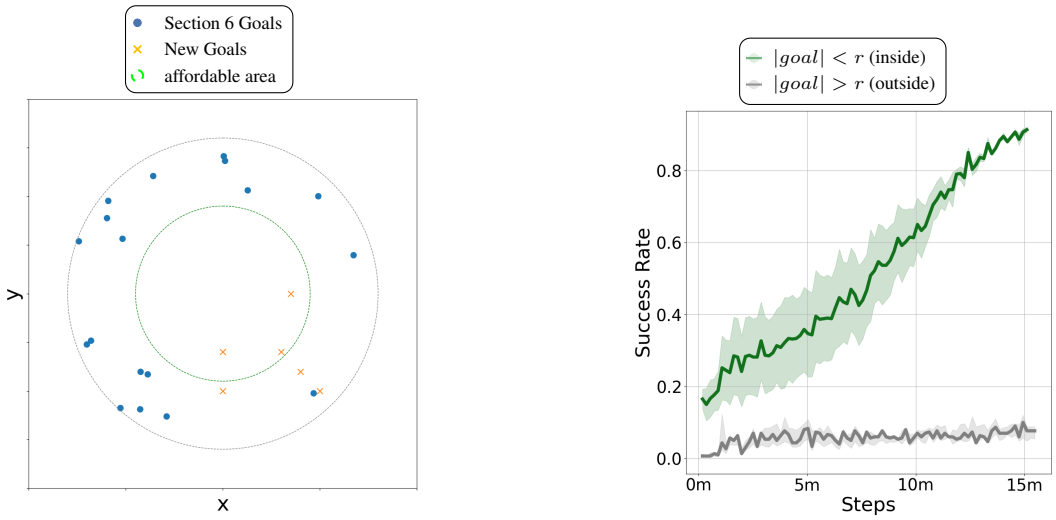


Figure 9: Parallel state entropy achieved by a single maximum entropy agent, against a collection of 10 or 50 agents, over longer runs. Mean State Entropy (solid line) and 95 % c.i. (shaded area)

DIAYN updates a skill discriminator $q(z \mid s)$ to maximize mutual information via categorical cross-entropy, whereas METRA optimizes a 2D metric representation $\phi(s)$ under a 1-Lipschitz constraint ($\lambda = 1.0$). Both auxiliary networks are trained using Adam with learning rate $\alpha_{\text{aux}} = 1 \times 10^{-4}$,

Table 5: Computation time across training stages. Times are reported as mean ± standard deviation.

Method	Average Time / Step (ms)	Average Policy Update Time / Update (s)
50 × Single Agent	30.07054 ± 0.81270	10.85052 ± 0.30234
50-Myriads	**2.41955 ± 0.00621**	**1.28916 ± 0.02304**



(a) Goal locations from Section 6 and new test goals for evaluating PPO capability.

(b) Median ± IQR success rate across 6 runs for PPO without pretraining

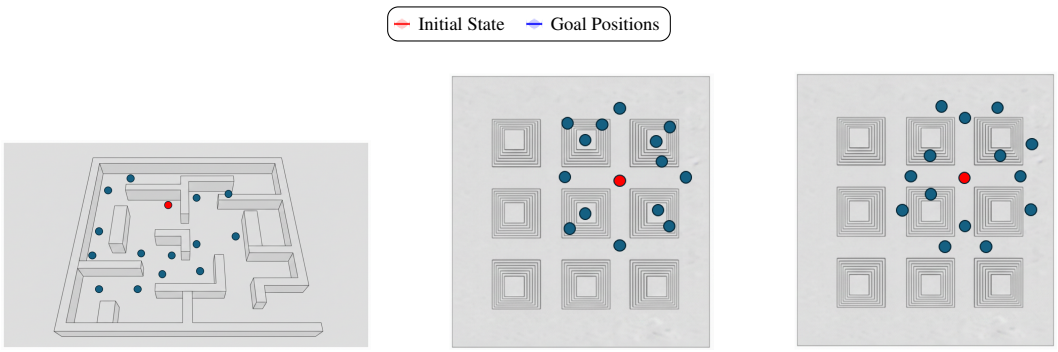
Figure 10: Comparison between goal region visualization and success rate performance for non-pretrained PPO agents. The left plot shows the spatial goal distribution with circular thresholds of radius r, where PPO without training achieves good performance, while the right plot shows the corresponding success rates over time.

while policies are optimized at $\alpha_\pi = 3 \times 10^{-4}$. The complete list of training and architectural hyperparameters is reported in Table 4.

D.2 Ablation on Pretraining and Agent Scaling

Convergence of Parallel agent entropy performance From Figure 9, we analyze the entropy evolution over an extended training run of 1,000 update steps in the *Empty* environment, which exhibits the most pronounced differences among the evaluated configurations. The single-agent setup continues to outperform the parallel-agent configurations, though the performance gap narrows over training. This trend supports the hypothesis that, given sufficient updates, the parallel configurations tend to converge toward the behavior of the single-agent case.

PPO Behavior Across Goal Regions The results in Figure 11 highlight a clear difference in the PPO agent's performance depending on the goal location. The agent demonstrates consistent improvement and achieves high success rates for goals located within the affordable region (i.e., $|goal| < r$) with a radius of $4.5m$. In contrast, the performance remains poor for the original goal set introduced in Section 6, corresponding to targets farther from the initial state ($|goal| > r$), even after extended training. This outcome indicates that PPO alone struggles to generalize to distant goals, and pretraining is therefore essential to effectively solve tasks that lie outside the agent's initial reachable area.



(a) Maze Environments (b) Cave Environments (c) Pyramid Environment

Figure 11: Example instances of the Maze, Cave, and Pyramid navigation environments. The agent's start state is marked in red, and candidate goal states in blue. Goal sampling is constrained to valid free-space regions to avoid placement inside walls

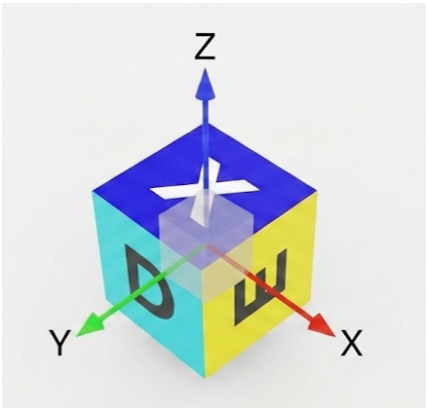


Figure 12: Cube orientation goals. The environment is trained using a set of six goal quaternions. Each goal corresponds to a successful manipulation task where a specific face is oriented upwards, serving as the target state for the reinforcement learning agent.

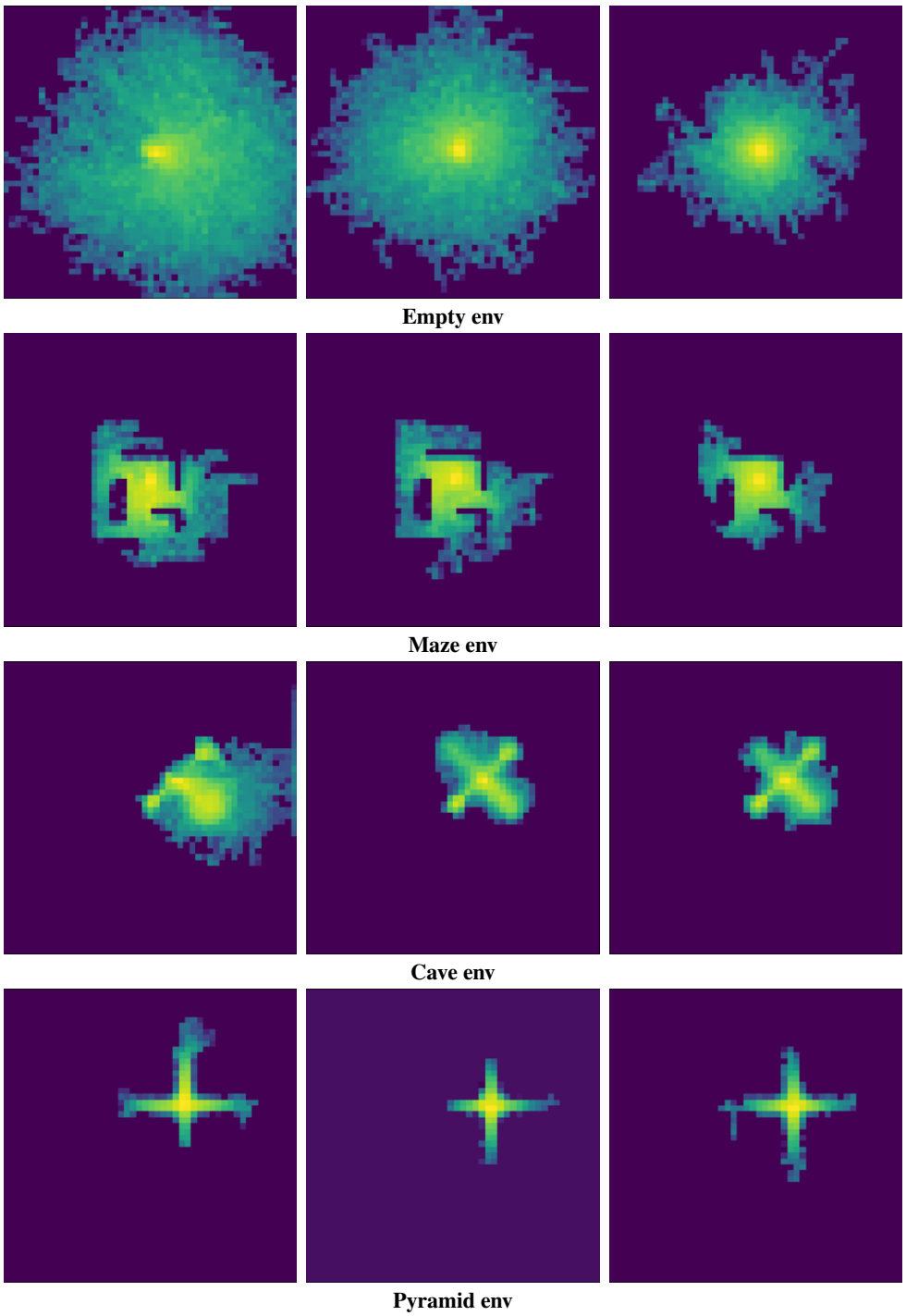


Figure 13: 2D heatmaps of the four environment configurations. Each row corresponds to a different environment, and each column shows the baseline, 10-agent, and 50-agent settings for the same random seed.