Unsupervised Action-Policy Quantization via Maximum Entropy Mixture Policies with Minimum Entropy Components

Yamen Habib
Department of Communication
and Information Technologies
Universitat Pompeu Fabra
Barcelona, Spain
yamen.habib@upf.edu

Dmytro Grytskyy
Department of Communication
and Information Technologies
Universitat Pompeu Fabra
Barcelona, Spain
dmytro.grytskyy@gmail.com

Rubén Moreno-Bote
Serra Húnter Fellow Programme
Department of Communication
and Information Technologies
Universitat Pompeu Fabra
Barcelona, Spain
ruben.moreno@upf.edu

Abstract

Most reinforcement learning approaches optimize behavior to maximize a task-specific reward. However, from this, it is difficult to learn transferable token-like behaviors that can be reused and composed to solve arbitrary downstream tasks. We introduce an online unsupervised reinforcement learning framework that autonomously quantizes the agent's action space into component policies via a joint entropy objective—maximizing the cumulative entropy of an overall mixture policy to ensure diverse, exploratory behavior under the maximum-occupancy principle, while minimizing the entropy of each component to enforce diversity and high specialization. Unlike existing approaches, our framework tackles action quantization into state-dependent component policies in a fully principled, unsupervised, online manner. We prove convergence in the tabular setting through a novel policy iteration algorithm, then extend to continuous control by fixing the discovered components and deploying them deterministically within an online optimizer to maximize cumulative reward. Empirical results demonstrate that our *maxi-mix-mini-com* entropy-based action-policy quantization provides interpretable, reusable token-like behavioral patterns, yielding a fully online, task-agnostic, scalable architecture that requires no task-specific offline data and transfers readily across tasks. Project Page: https://yamenhabib.com/maxi-mix-mini-com/ .

1 Introduction

Continuous control tasks in biophysical systems and robots require mapping high-dimensional state inputs to actions in real time [1]. This presents significant challenges: the vastness of the continuous action space makes exploration inefficient [2], and balancing the exploration–exploitation trade-off often necessitates approximations that complicate policy optimization [3]. Action quantization—discretizing continuous actions via a finite number of actions—has been explored as a means to

18th European Workshop on Reinforcement Learning (EWRL 2025).

address the difficulties of continuous control [4–6]. By focusing on core representative actions, action quantization can improve sample efficiency, accelerate convergence, and avoid wasteful exploration of irrelevant continuous actions [7]. Beyond its technical strengths, action quantization can allow simpler labeling and interpretability by human observers, making agent behavior easier to understand compared to continuous actions. It can also simplify planning or sequence generation by arbitrarily composing token-like actions [8, 9].

Despite its theoretical interest, action quantization presents its own fundamental challenges. First, naïvely binning uniformly across each action dimension leads to an exponential growth of action space—resulting in K^M total bins for M dimensions and K levels each—which quickly becomes computationally intractable as dimensionality increases. Some approaches treat each action dimension independently to avoid the curse of dimensionality [5], but this approximation overlooks the temporal correlations across dimensions inherent in realistic action sequences (e.g., as in multi-joint control tasks) (see Appendix 6.1 for further discussion). Bin placing in action space is another related problem, whose solution requires action quantifiers to become state-dependent [6, 7] and task-dependent [10]. The second fundamental challenge is how to learn action quantizers that are task-agnostic and transferable across tasks. Current approaches rely on curated offline datasets to allow for exhaustive coverage of the state–action space, but this is impractical in dynamic, on-robot deployments [7]. More importantly, exclusively relying on offline data ties the quantization scheme to the dataset's task, and fixed, task-specific bins must be fully retrained for each new environment, hindering both generalization and reusability [6]. Further, existing work often lacks a principled analysis of quantization, focusing primarily on reward maximization without systematically studying the theoretical properties, reusability, or long-term behavior of the resulting discrete action sets (see Appendix 6.1). In contrast to the dominance of task-specificity in reinforcement learning (RL), unsupervised pre-training has emerged as a fundamental learning framework, generating significant advances in both natural language processing and computer vision. In language, generative autoregressive models [11] trained on large unlabeled text corpora learn versatile representations that can be fine-tuned for diverse downstream tasks; in vision, contrastive learning methods [12] derive discriminative feature embeddings from unannotated images. In this work we address the above two challenges: how to build in an unsupervised manner state-dependent, token-like quantized action-policies that are transferable and easily composable to solve any downstream task.

Using a Markov Decision Process (MDP) setting, we reframe action quantization as the problem of learning a finite set of component policies each with minimal action entropy, whose combination into a mixture is optimized to maximize cumulative action entropy. By jointly optimizing these opposing objectives—high entropy mixture with minimal component entropy—the mixture policy largely samples action space in a task-agnostic manner while identifying and excluding risk-prone action regions that would hinder future action entropy. In parallel, each state-dependent component refines its action repertoire to execute finely tuned maneuvers when invoked, and they become diverse because they must jointly produce a maximum entropy mixture policy. This *maxi-mix-mini-com entropy* formulation for action-policy quantization is entirely unsupervised and does not rely on any offline dataset or task-specific demonstrations; because quantization arises from the agent's intrinsic motivation to generate safe action entropy in the long term—*the maximum occupancy principle* (MOP) [13, 14])— rather than solving a task, the resulting mixture components provide reusable tokens that are transferable across different tasks and environments.

We first show that the maxi-mix-mini-com entropy objective is tractable, and we provide a provably convergent policy-iteration algorithm for the discrete setting (Sec. 2). Building on the tabular case, we derive and validate a policy-gradient theorem under function approximation to extend our framework to continuous actions (Fig. 1). The outcome of the joint entropy objective is a set of component policies with support on continuous action; these components can be deployed by themselves to allow for smooth composition of token-like behaviors. However, we find that simply executing components using their modes to maximize arbitrary reward functions leads to great transferability. Through experiments, we show that learned components scale effectively, transferring successfully to novel environment layouts and achieving performance comparable to leading baselines such as SAC [15] and PPO [16] in high-dimensional, on-policy control tasks (Sec. 4). Because components are acquired in a fully unsupervised fashion—driven solely by the agent's intrinsic objective—they remain agnostic to any specific reward function. Consequently, this quantization alone cannot, in principle, recover the exact optimal continuous action for an arbitrary, previously unseen reward signal. Nonetheless, we find that as the number of learned components provides a good enough

coverage of action space the mixture policy attains performance on par with leading continuous-action baselines (Sec. 4). Our results establish a principled framework for action-policy quantization resulting in interpretable components that can be flexibly composed in a token-like manner for efficient exploration and generalizes well to unseen tasks.

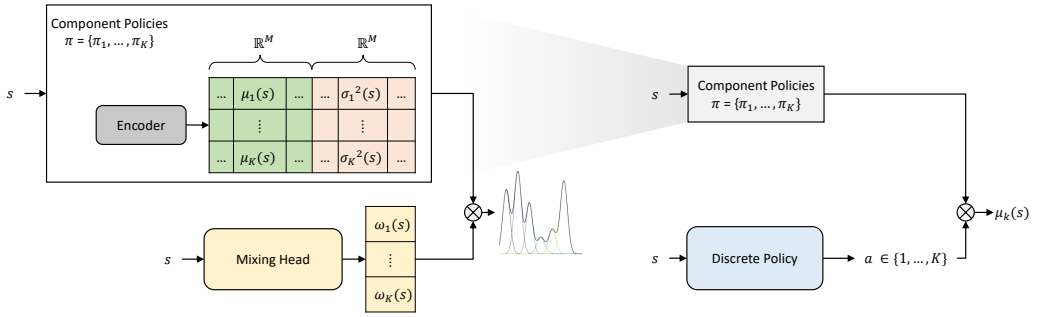

Figure 1: Unsupervised *maxi-mix-mini-com* entropy learning and deployment in the continuous case. **(Left)** Unsupervised pre-training delivers a set of K component policies $\{\pi_k\}$ forming a diverse repertoire of behaviors, each with mean $\mu_k(s)$ and diagonal covariance $\sigma_k^2(s)$)(see Appendix 6.5 for more details). **(Right)** For downstream tasks, these components are frozen and treated as black-box behavioral tokens for any discrete-action algorithm (e.g., DQN, PPO). The discrete action space is the index set $\{1, \ldots, K\}$, where selecting index k executes the corresponding mode action $\mu_k(s)$ (or, optionally, a low-variance sample) in the environment to accumulate task-specific reward.

2 Maxi-mix-mini-com entropy objective for mixture component learning

Our goal is to learn a mixture policy and its components so that the mixture generates maximum cumulative entropy under MOP while each component has minimal entropy. We address this problem in the unsupervised setting, where the agent is not faced with any task. We first start in the discrete action-state case, and later address the continuous action-state case (Sec. 3). We consider a Markov decision process (MDP) $(\mathcal{S}, \mathcal{A}, p, \gamma)$, where $\mathcal{S}$ is a discrete state space, $\mathcal{A}$(s) is a state-dependent discrete action space, $p(s'|s, a)$ is the transition kernel, and $\gamma \in [0, 1)$ is the discount factor.

We define the mixture policy

$$\pi_m(a|s) = \sum_{k=1}^{K} w_k(s)\pi_k(a|s) \tag{1}$$

as a non-negative linear combination of K component policies $\pi_k(a|s)$ with mixture weights $w_k(s) \geq 0$ and $\sum_k w_k(s) = 1$. Weights are state-dependent, i.e., they are not fixed, and they can be interpreted as the probability that policy k is chosen at state s. The number of components K can be made state-dependent, $K(s)$, in a straightforward manner.

Our goal is to optimize both the component policies and weights so as to maximize the value function

$$V(s) = \sum_{t=0}^{\infty} \gamma^t \mathbb{E}\left[\mathcal{H}(\pi_m(\cdot|s_t)) - \alpha \sum_k w_k(s_t)\mathcal{H}(\pi_k(\cdot|s_t))\middle| S_0 = s\right], \tag{2}$$

where the expectation is over actions sampled from the mixture policy and state transitions given initial condition s, $\mathcal{H}(\pi(\cdot|s)) = -\sum_a \pi(a|s)\log\pi(a|s)$ is the action entropy of a policy π given state s_t at time step t, and $0 \leq \alpha < 1$. This goal maximizes the sum of future action entropies, thus generating diversity of action paths, while minimizing the entropy of each of the components, thus favoring specialized component policies. These two generally opposing goals are balanced by the hyperparameter α. The degree of diversity thus generated is also shaped by the state-dependence of the action set $\mathcal{A}(s)$. Indeed, to avoid the possibility of the trivial solution of a uniform policy as the optimal mixture, we assume that the environment is embedded with some structure, typically with terminal states s^+ for which only a single action is allowed $|\mathcal{A}(s^+)| = 1$, which is "not doing anything". State-dependence actions sets strongly shape the solution, leading to maximum entropy policies that promote exploration and diverse behavior while trying to avoid low-size action states,

like terminal states (e.g., such a deadly falling down) from where no further action entropy can be generated. Finally, we note that replacing the weighted sum of component policy entropies in Eq. 2, $-\sum_k w_k(s_t)\mathcal{H}(\pi_k(\cdot|s_t))$, by an unweighted one, $-\sum_k \mathcal{H}(\pi_k(\cdot|s_t))$ would penalize as well having entropic component policies, but it would not lead to an easily solvable problem (see Appendix 6.2.1).

Eq. 2 can be recursively expressed in the form of the Bellman equation

$$V(s) = \mathcal{H}(\pi_m(\cdot|s_t)) - \alpha \sum_k w_k(s_t)\mathcal{H}(\pi_k(\cdot|s_t)) + \gamma \sum_a \pi_m(a|s)Q(s,a) \tag{3}$$

$$Q(s,a) \equiv \sum_s p(s'|s,a)V(s) \; . \tag{4}$$

Therefore, the objective can be expressed as maximizing the value function over components and mixture weights as the optimality Bellman equation

$$V^*(s) = \max_{\pi,w} \left(R(s;\pi,w) + \gamma \sum_a \pi_m(a|s)Q^*(s,a) \right) \tag{5}$$

$$R(s;\pi,w) \equiv \mathcal{H}(\pi_m(\cdot|s)) - \alpha \sum_k w_k(s)\mathcal{H}(\pi_k(\cdot|s)) \; , \tag{6}$$

where π indicates the set of component policies $\pi = \{\pi_1, ..., \pi_K\}$ and $w = \{w_1, ..., w_K\}$, where the conditioning on the state s is omitted but understood.

We first prove that for a high capacity mixture policy (i.e., when the number of components is larger or equal than action space size) solving Eq. 5 is equivalent to solving an unconstrained problem with no restrictions on the mixture policy. This result, although somehow obvious, shows that there is a limit where the problem converges to a known solution.

Theorem 1. *If $K(s) \geq |\mathcal{A}(s)|$ for all s, then the optimal solution of Eq. 5 is such that $\pi_k(a_k|s) = 1$ only for one action a_k and the set of actions spans $\mathcal{A}(s)$, that is, $\{a_k\} = \mathcal{A}(s)$. Then, π_m in Eq. 1 is unconstrained and the optimal value function obeys*

$$V^*(s) = \max_{\pi,w} \left(\mathcal{H}(\pi_m(\cdot|s_t)) + \gamma \sum_a \pi_m(a|s)Q^*(s,a) \right) \; . \tag{7}$$

The optimal mixture policy is unique, while the mixture components are not unique if $|\mathcal{A}(s)| > 1$ at least for one state.

Proof. See Appendix Sec. 6.2. While the case $K(s) \geq |\mathcal{A}(s)|$ is interesting, the most relevant scenario occurs when the number of components is smaller than the action dimensionality, $K(s) < |\mathcal{A}(s)|$, as this will lead to a compression and quantization of action space into components that can later be used to compose sequences of actions for reward maximization using, e.g., off-the-shelf DQN [17], or for efficient planning [9]. If mixture capacity is low, directly optimizing Eq. 5 is hard because derivatives with respect to both π and w lead to a set of nonlinear questions that to our knowledge cannot be solved explicitly (Sec. 6.2). To bypass this problem, we derive an iterative algorithm, which is the main theoretical contribution of our paper.

We first introduce the responsibility

$$r_k^*(s,a) = \frac{w_k(s)\,\pi_k(a|s)}{\pi_m(a|s)} \; , \tag{8}$$

which is the probability that a performed action a at state s has been generated from the component policy k – note that the denominator, defined in Eq. 1, makes $r_k^*(s,a)$ probabilities for all (s,a), that is, $r_k^*(s,a) \geq 0$ and $\sum_k r_k^*(s,a) = 1$. Next, we define the immediate gain $G(s;\pi,w,r)$ as

$$G(s;\pi,w,r) = -\sum_k w_k(s)\log w_k(s) - (1-\alpha)\sum_{k,a} w_k(s)\,\pi_k(a|s)\log \pi_k(a|s) \tag{9}$$

$$+ \sum_{k,a} w_k(s)\pi_k(a|s)\log r_k(s,a) \, , \tag{10}$$

where $r = \{r_1, ..., r_K\}$ is a probability vector. It can be easily seen that (i) $G(s;\pi,w,r^*) = R(s;\pi,w)$ when using the probability vector r^* given by Eq. 8, and (ii) $G(s;\pi,w,r^*) \geq$

$G(s;\pi,w,r)$ for any probability vector r [18]. The above definitions and observations allow us to write the optimality Bellman equation 5 as

$$V^*(s) = \max_{\pi,w} \max_{r} \left(G(s;\pi,w,r) + \gamma \sum_a \pi_m(a|s) Q^*(s,a) \right) . \tag{11}$$

As in the Blahut–Arimoto algorithm [19, 18], we use this fact to build a algorithm that leads to monotonic improvement of the value function until convergence.

Our algorithm, shown in Algorithm 1, proceeds as follows:

(1) Initial conditions: Start the value function $V^{(0)}(s) = 0$ for all s and arbitrary $\pi^{(0)} = \{\pi_1^{(0)}, ..., \pi_K^{(0)}\}$ and $w^{(0)} = \{w_1^{(0)}, ..., w_K^{(0)}\}$ outside the simplex boundaries, that is, $\pi_k^{(0)}(a|s) > 0$ and $w_k^{(0)}(s) > 0$ for all s and a. Define the initial responsibilities as

$$r_k^{(0)}(s,a) = \frac{w_k^{(0)}(s)\, \pi_k^{(0)}(a|s)}{\pi_m^{(0)}(a|s)} > 0 . \tag{12}$$

Iterate the following step (2) for $n = 1, 2, ...$

(2) At iteration n, start from $V^{(n-1)}$, $\pi^{(n-1)}$, $w^{(n-1)}$ and $r^{(n-1)}$ and perform steps (2a)-(2d):

(2a) Define the updated value function $V^{(n)}$ as

$$V^{(n)}(s) = G(s;\pi^{(n-1)}, w^{(n-1)}, r^{(n-1)}) + \gamma \sum_a \pi_m^{(n-1)}(a|s) Q^{(n-1)}(s,a) , \tag{13}$$

with $Q^{(n-1)}(s,a) = \sum_{s'} p(s'|s,a) V^{(n-1)}(s)$ and $\pi_m^{(n-1)} = \sum_k w_k^{(n-1)}(s) \pi_k^{(n-1)}(a|s)$.

(2b) Compute $\pi^{(n)} = \{\pi_1^{(n)} \cdots, \pi_K^{(n)}\}$ fixing $V^{(n-1)}$ and $r^{(n-1)}$ using

$$\pi_k^{(n)}(a|s) = \frac{[r_k^{(n-1)}(s,a)]^{\frac{1}{1-\alpha}} e^{\frac{\gamma}{1-\alpha} Q^{(n-1)}(s,a)}}{\sum_{a'} [r_k^{(n-1)}(s,a')]^{\frac{1}{1-\alpha}} e^{\frac{\gamma}{1-\alpha} Q^{(n-1)}(s,a')}} . \tag{14}$$

(2c) Compute $w^{(n)}$ fixing $V^{(n-1)}$ and $r^{(n-1)}$ as

$$w_k^{(n)}(s) = \frac{\left(\sum_a [r_k^{(n-1)}(s,a)]^{\frac{1}{1-\alpha}} e^{\frac{\gamma}{1-\alpha} Q^{(n-1)}(s,a)} \right)^{1-\alpha}}{\sum_l \left(\sum_a [r_l^{(n-1)}(s,a)]^{\frac{1}{1-\alpha}} e^{\frac{\gamma}{1-\alpha} Q^{(n-1)}(s,a)} \right)^{1-\alpha}} . \tag{15}$$

(2d) Compute $r^{(n)}$ fixing $V^{(n-1)}$, $\pi^{(n)}$ and $w^{(n)}$ using

$$r_k^{(n)}(s,a) = \frac{w_k^{(n)}(s)\, \pi_k^{(n)}(a|s)}{\pi_m^{(n)}(a|s)} , \tag{16}$$

with $\pi_m^{(n)}(a|s) = \sum_k w_k^{(n)}(s)\, \pi_k^{(n)}(a|s)$.

Theorem 2. *The algorithm following steps (1) and (2), which consists in iterating the π, w and r using Eqs. 13,14,15,16 in this order, improves the value function monotonically until convergence.*

Proof. See Appendix Sec. 6.2.

2.1 Experiments and results

To study how component policies are learned in the discrete action case when the capacity of the policy is smaller than the size of the action space, $K(s) < |\mathcal{A}(s)|$, we run some experiments in a fully deterministic grid-world of impassable walls and terminal cells. The action set is {`up`, `right`, `down`, `left`} when away from the walls. Any action that would move into a wall is not available, leading to a state-dependent action set that is reduced to 3 (or 2 in case of a corner) actions when next to a wall. Actions leading into a terminal cell remain available and immediately terminate the episode

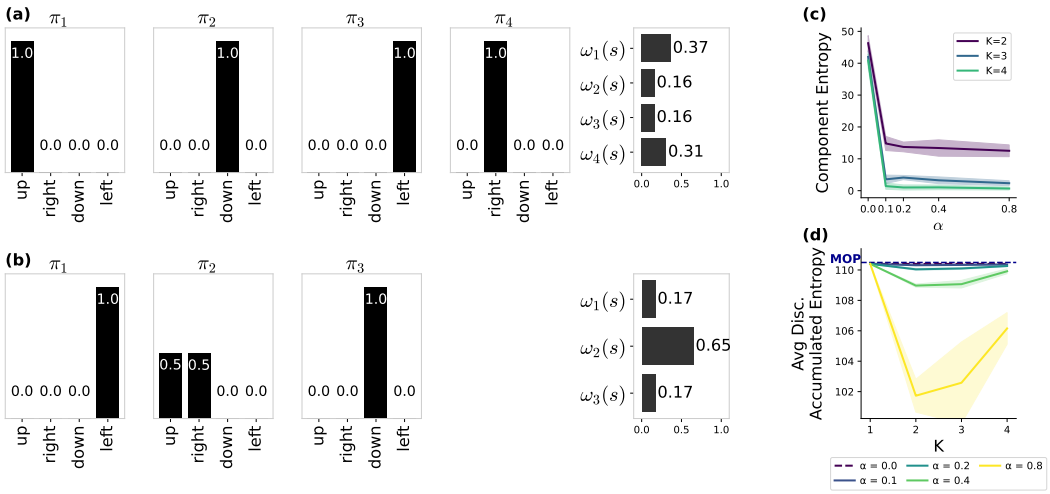

Figure 2: Component specialization and cumulative mixture entropy in a grid world (Fig. 4, Appendix 6.4). **(a,b)** Learned state-dependent policies when $K = 4$ (a) and $K = 3$ (b). Policies $\pi_1, \pi_2, \ldots$ (left panels) and mixture weights $w_k(s)$ (right panel) are shown for state $s = 141$ in the grid-world. **(c)** Averaged component entropy decreases as a function of α and for increasing K (different lines), showing enhanced specialization. **(d)** Cumulative mixture entropy increases with the number of components K and for higher α (different lines). Solid curves: mean performance over five random seeds; shaded regions: their variance. Dashed line in panel (d) indicates MOP, the theoretical limit.

if taken, which is equivalent to having a single action and producing no entropy from there onward. Because our focus is purely unsupervised, no extrinsic rewards are provided: learning is driven by the maxi-mix-mini-com entropy objective. To obtain the optimal mixture policy and components we use Algorithm 1, described in Sec. 2.

We find that when $K = 4$, component policies are sharply peaked at a single action, and they cover all action sets (Fig. 2a), as predicted by Theorem 2. When capacity is smaller (e.g., $K = 3$), some components become sharp, while others are not (Fig. 2b), but all of them together offer diversity and cover the full action space. Components are state-dependent (not shown). In general, we observe that the policy components with higher weight tend to be broader than the ones that are chosen less frequently (see second component in Fig. 2b and its associated weight). This bias is consistent with the maxi-mix-mini-com entropy objective: if mostly only one component policy is active, it should produce high action entropy to be in accord with the max-mix objective part. However, the agent generates safe component policies in the sense that the probability of falling to a terminal state is zero or almost zero, a fact that strongly shapes the final mixture policy. As expected, the entropy of the component policies declines with increased α and when increasing the number of components K (Fig. 2c), enhancing their specialization. At the same time, the cumulative entropy of the mixture policy increases with K regardless of α (Fig. 2d) and approaches the theoretical limit imposed by MOP (dashed line; MOP is equivalent to $\alpha = 0$ and $K = 1$). When $K = 1$, the problem is equivalent to MOP for all α.

3 Component learning in continuous case

Our discrete grid-world experiments establish theoretical feasibility and validate the maxi-mix-mini-com entropy objective. Now, we extend it to the continuous-action setting. First, we formalize the continuous-action problem and present a training procedure for Gaussian mixture policies (see Fig. 1). Next, we analyze the resulting components, quantifying the framework's ability to control their coverage and diversity. Because the continuous-action joint-entropy objective is entirely unsupervised (no extrinsic rewards are available), the standard policy-gradient theorem does not apply directly, but we can adapt it to our objective function:

Theorem 3 (Extended Policy Gradient Theorem)**.** *Let $\pi_{m,\theta}(a|s)$ be a mixture policy over continuous actions, and define the state-value function as in Eq. 3. Then the gradient of the performance objective $J(\theta) = V(s_0)$ admits the form*

$$\nabla_\theta J(\theta) = \int_{s,a} d_{\pi_{m,\theta}}(s)\pi_{m,\theta}(a|s)\Big(\nabla_\theta \log \pi_{m,\theta}(a|s)Q(s,a) + \nabla_\theta R(s;\pi_\theta, w_\theta)\Big)\, da\, ds.$$

where $\nabla_\theta R(s;\pi_\theta, w_\theta) = \nabla_\theta \mathcal{H}\big(\pi_{m,\theta}(\cdot|s)\big) - \alpha\nabla_\theta \sum_{k=1}^{K} w_{k,\theta}(s)\, \mathcal{H}(\pi_{k,\theta}(\cdot|s))$, $\pi_\theta = \{\pi_{1,\theta}, ..., \pi_{K,\theta}\}$ *and* $w_\theta = \{w_{1,\theta}, ..., w_{K,\theta}\}$

Proof. See Appendix Sec. 6.3.

There are several methods for optimizing the policy objective $J(\theta)$ using policy gradient RL. One common approach is the likelihood-ratio gradient estimator [20], which does not require backpropagation through the Q-function or environment dynamics, as it relies on sampled returns. In our setting, the Q-function is represented by a differentiable neural network, enabling the use of the reparameterization gradient estimator for each Gaussian policy component. This estimator is unbiased, achieves lower variance than the likelihood-ratio approach by leveraging deterministic gradient computations through both the policy and Q-function networks [15, 21], and adapts effectively to our context.

Here, the component policies are reparametrized as $\pi_{k,\theta}(a|s) = p(\varepsilon)$ where $a = f_\theta(\varepsilon; s, k)$ and $p(\cdot)$ is the noise distribution. We formalize this in the Component-Reparameterization Policy Gradient Theorem 4, providing an unbiased gradient form for optimization.

Theorem 4 (Component-Reparameterization Policy Gradient Theorem)**.** *Let $\hat{f}_\theta = f_\theta(\varepsilon; s, k)$, then the gradients of objective $\nabla_\theta J\big(\pi_{m,\theta}\big)$ function under our reparametrization become*

$$\begin{aligned}\mathbb{E}_{\substack{s\sim d_{\pi_{m,\theta}} \\ k\sim Cat(w_{k,\theta}(s)) \\ \varepsilon\sim p}}\Bigg[&\nabla_\theta \log w_{k,\theta}(s)\,\Big(Q_{\pi_{m,\theta}}\big(s,\hat{f}_\theta\big) - \log \pi_{m,\theta}\big(\hat{f}_\theta|s\big) + \alpha \log \pi_{k,\theta}\big(\hat{f}_\theta|s\big)\Big) \\ &+\ \nabla_\theta \hat{f}_\theta\ \nabla_a\Big(Q_{\pi_{m,\theta}}(s,a) - \log \pi_{m,\theta}(a|s) + \alpha \log \pi_{k,\theta}(a|s)\Big)\Big|_{a=\hat{f}_\theta}\Bigg],\end{aligned} \tag{17}$$

where $d_{\pi_{m,\theta}}(s)$ is the discounted occupancy measure under $\pi_{m,\theta}$.

Proof. See Appendix Sec. 6.3.

Since the mixture weights $w_\theta(\cdot)$ cannot be directly reparameterized, we adopt the straight-through Gumbel–Softmax trick from [22] to produce a biased but differentiable sample. First, draw $g_1, \ldots, g_K \overset{\text{i.i.d.}}{\sim} \text{Gumbel}(0,1)$ and form the "soft" sample

$$y_k = \frac{\exp\big((\log w_{k,\theta}(s) + g_k)/\tau\big)}{\sum_{j=1}^{K}\exp\big((\log w_{j,\theta}(s) + g_j)/\tau\big)}, \quad k = 1, \ldots, K,$$

where τ is the temperature parameter controlling the smoothness of the distribution. A discrete sample is obtained as

$$\hat{z} = \text{one\_hot}\Big(\arg\max_k(\log w_{k,\theta}(s) + g_k)\Big), \quad [\text{one\_hot}(i)]_k = \delta_{k,i}.$$

Applying the straight-through estimator yields

$$z_k = \hat{z}_k + y_k - \text{StopGrad}(y_k), \quad \text{StopGrad}(y) = y,\ \nabla_y \text{StopGrad}(y) = 0,$$

so that gradients flow through y_k but not through its stopped version. Finally, combining $z = (z_1, ..., z_K)$ with reparameterized samples $f_\theta(\varepsilon; s, k)$ from each component gives $a = \sum_k z_k\, f_\theta(\varepsilon; s, k)$.

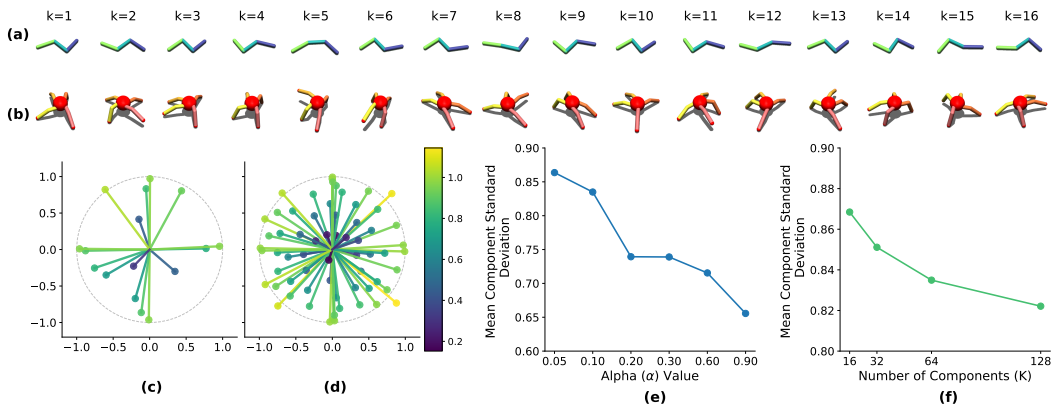

Figure 3: **(a)** Mujoco Swimmer-V5 and **(b)** Ant-V5 (bottom) poses resulted from continuously applying the mode of each learned component policy as action, showing diversity and coverage of action space. **(c)** Swimmer ($K = 16$) mode vectors. **(d)** Swimmer ($K = 64$) showing uniform torque coverage. **(e)** Ant ($K = 64$) mean component std vs. α. **(f)** Ant mean std vs. K.

3.1 Experiments and results

We first used the MuJoCo Swimmer benchmark [23], which comprises three segments connected by two rotary joints in a linear chain, and with no terminal states. Our algorithm learns a set of components that are readily interpretable as stretches and contractions, of different sorts, of the body (Fig. 3a, for $K = 16$; see Fig. 7 Appendix 6.6 for $K = 64$; Videos). Their diversity and coverage are better visualized by plotting the modes of each of the Gaussian components, using 16 components (Fig. 3c) or 64 components (Fig. 3d). With more components, the coverage is more uniform in both angles and strength, so that the components can be interpreted as torques in all possible directions and with different magnitudes (colors).

We evaluated maxi-mix-mini-com on the MuJoCo Ant benchmark [23], a three-dimensional quadruped with a central torso and four two-segment legs (eight actuated hinge joints). We consider terminal states as those cases where the torso touches the floor, consisting in ant falls. Like the swimmer, the Ant learns component policies resulting in interpretable coordinated movements when deployed using their modes (Fig. 3b, for $K = 16$; see Fig. 8 Appendix 6.6 for $K = 64$; Videos). The components are diverse and cover a broad range of reaching movements with one leg, in addition to adopting various bow poses, while keeping a balance to avoid falling down to the floor. As components are Gaussian, we use their standard deviation to characterize their specialization as a function of the number of components K and the regularizer α. The standard deviation averaged over components when $K = 64$ decreases with α (Fig. 3e), indicating an increased degree of component specialization. Similarly, increasing the number of components reduces component standard deviation (and thus their entropy) (Fig. 3f). These observations are in line with the expectations of how the specialization of the components should depend on the number of components and on the regularizer.

4 Hierarchical reinforcement learning with pre-trained components

As demonstrated in the previous section, our method generates diverse yet specialized component policies. We now use the learned components to test their effectiveness and reusability in solving different tasks by measuring cumulative reward (see Fig. 1). Action space is discretized using the modes of the learned component Gaussians, and use them as discrete actions in a DQN [17] or discrete PPO [16]. The learned components and the associated discretization are not allowed to change during the optimization of cumulative reward using the discrete algorithm (e.g. DQN); only the hierarchical policy on top of them is optimized. Because the downstream controller uses the fixed finite set of component modes $\mathcal{A}_K(s) = \{\mu_k(s)\}_{k=1}^K \subset \mathcal{A}(s)$, any policy constrained to $\mathcal{A}_K(s)$ is upper-bounded by one that optimizes over the full continuous action set. Accordingly, when a single set of K components must support N heterogeneous rewards with $K \ll N$, a reduction in achievable return relative to continuous-action methods (e.g., SAC [15]) is expected, unless the continuous optima are contained in or well-approximated by $\mathcal{A}_K(s)$. Finally, random action quantization fails to

capture the coordinated, state-dependent joint couplings required in high-dimensional control, so a small randomly sampled codebook is unlikely to contain either stabilizing torques or actions suitable for downstream tasks, as our preliminary experiments indicate (not shown).

Table 1: Mean per-episode reward (±SE) on Ant locomotion tasks using same 64 fixed components.

Method	Directional Avg.	Maze
SAC (continuous)	3770 ± 42.23	11.48 ± 0.36
PPO (continuous)	221 ± 11.12	3.42 ± 0.31
Ours ($K = 64$ discrete)	2118 ± 25.47	9.6 ± 0.28

We first used two very different locomotion tasks on the MuJoCo Ant: maximizing torso velocity along one of the 4 cardinal directions (directional locomotion task, with four conditions), and reaching a central goal in a maze from a random perimeter start (maze navigation task). These two tasks a priori need different skills, such as running (in the directional locomotion task) and steering the torso and turning in different directions (in the maze navigation tasks). We compared the performance of our algorithm against Soft Actor–Critic (SAC) [15] and standard continuous PPO [16] (Table 1; see Appendix 6.5 for training details of each algorithm). With only $K = 64$ discrete components, our method recovers 56% of SAC's performance and outperforms continuous PPO by nearly an order of magnitude on the directional locomotion tasks. Furthermore, using these same components, its performance remains comparable to the continuous baselines in the maze navigation task, indicating that the learned components generalize effectively across different locomotion challenges. We emphasize that this result is quite remarkable because the components have not been specialized or fine tuned to any of the two tasks: the maxi-mix-mini-com entropy algorithm learn components that are reusable, diverse enough to solve different tasks, and flexible enough to be composed to generate complex sequential behavior.

Table 2: Mean per-episode reward (±SE) on manipulation and locomotion tasks.

Method	Fetch Reach (success rate)	Swimmer	Hopper	Walker2D
SAC (continuous)	0.84 ± 0.07	110.54 ± 1.52	2884.79 ± 158.22	4734.71 ± 102.76
PPO (continuous)	0.70 ± 0.16	70.53 ± 0.89	2647.90 ± 120.96	3488.76 ± 63.26
Ours ($K = 64$ discrete)	1.00 ± 0.00	117.57 ± 1.17	2332.12 ± 98.70	1024.62 ± 112.53

We also tested in domains with smaller action spaces (Fetch Reach and Swimmer) to determine whether a denser action coverage leads to a comparably better performance. Indeed, in these cases the maxi-mix-mini-com entropy objective not only matches but surpasses state-of-the-art continuous methods (Table 2). Further, for the more complex Hopper task—characterized by richer dynamics and higher-dimensional control—our approach recovers over 80 % of SAC's performance, illustrating robustness under increased complexity. Degradation of performance continued with increased action complexity: on Walker2D—an asymmetric, 6-dimensional forward-locomotion task—our method attains 21.8 % of SAC's performance, but still keeping the advantage of usability and transferability of the learned components, which could be further complemented by a larger number of components.

5 Discussion and Conclusion

We have presented an online unsupervised method to discover discrete action-policies using the maxi-mix-mini-com entropy objective. This objective promotes the generation of complex and diverse behaviors because the mixture policy is trained to maximize its future cumulative entropy, which entails generating complex, yet safe, behavioral patterns [13, 14]. The objective also promotes the discovery of distinct and specialized discrete action-policies because of the per-component entropy penalty.

Our framework tackles a very important problem that has been overlooked: how to generate state-dependent token-like actions that are transferable to multiple tasks. This challenge requires that data become available online, as opposed to being offline and fixed. Indeed, offline data from data-self-play [6] or human demonstrations [7] can facilitate discrete action discovery and initially speed up learning. However, in the long term, a fixed offline dataset curtails the possibility of the agent to freely discover new behaviors, especially hurting the generation of a diverse set of action-policies, which we showed

above to be critical for reusability and transferability. At the same time, online, unsupervised learning in our maxi-mix-mini-com entropy algorithm is significantly harder than offline learning: the agent must learn by itself to generate diverse action-policies that ensure smooth transitions between them to maintain stability and avoid catastrophic failures. Despite these complexities, our method achieves strong performance on tasks like maze and Fetch Reach, with high scores relative to the baseline, and experiments on Ant, Hopper, and Walker demonstrate its ability to find scalable, near-optimal solutions, highlighting its robustness.

Limitations: Our online learning method faces practical challenges. Notably, we often need to downscale the variance of the learned component policies to make them compatible with discrete actions. In some instances, we also resort to taking the modes of these components. However, this is not an inherent flaw in our algorithm—by increasing the number of components K, the standard deviation naturally decreases, rendering the policies more suitable for discrete policy reward maximization. Furthermore, downscaling is a technique also utilized in concurrent work, such as [9], to mitigate similar challenges. A further limitation concerns the trade-off between deployment efficiency and training overhead. Although at test time our framework can execute only a single component—yielding fast inference—its unsupervised learning phase requires evaluating all K component policies to compute the mixture's entropy. As K grows, this per-step entropy calculation incurs an increase in computation, making training slower for larger component sets.

References

[1] Daniel M Wolpert, Zoubin Ghahramani, and Michael Jordan. Forward dynamic models in human motor control: Psychophysical evidence. *Advances in neural information processing systems*, 7, 1994.

[2] 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. *arXiv preprint arXiv:1509.02971*, 2015.

[3] Murtaza Dalal, Deepak Pathak, and Russ R Salakhutdinov. Accelerating robotic reinforcement learning via parameterized action primitives. *Advances in Neural Information Processing Systems*, 34:21847–21859, 2021.

[4] Yuanyang Zhu, Zhi Wang, Yuanheng Zhu, Chunlin Chen, and Dongbin Zhao. Discretizing continuous action space with unimodal probability distributions for on-policy reinforcement learning. *IEEE Transactions on Neural Networks and Learning Systems*, 2024.

[5] Yunhao Tang and Shipra Agrawal. Discretizing continuous action space for on-policy optimization. In *Proceedings of the aaai conference on artificial intelligence*, volume 34, pages 5981–5988, 2020.

[6] Jianlan Luo, Perry Dong, Jeffrey Wu, Aviral Kumar, Xinyang Geng, and Sergey Levine. Action-quantized offline reinforcement learning for robotic skill learning. In *Conference on Robot Learning*, pages 1348–1361. PMLR, 2023.

[7] Robert Dadashi, Léonard Hussenot, Damien Vincent, Sertan Girgin, Anton Raichuk, Matthieu Geist, and Olivier Pietquin. Continuous control with action quantization from demonstrations. *arXiv preprint arXiv:2110.10149*, 2021.

[8] Yevgen Chebotar, Quan Vuong, Karol Hausman, Fei Xia, Yao Lu, Alex Irpan, Aviral Kumar, Tianhe Yu, Alexander Herzog, Karl Pertsch, et al. Q-transformer: Scalable offline reinforcement learning via autoregressive q-functions. In *Conference on Robot Learning*, pages 3909–3928. PMLR, 2023.

[9] Ziyad Sheebaelhamd, Michael Tschannen, Michael Muehlebach, and Claire Vernade. Quantization-free autoregressive action transformer. *arXiv preprint arXiv:2503.14259*, 2025.

[10] Younggyo Seo, Jafar Uruç, and Stephen James. Continuous control with coarse-to-fine reinforcement learning. *arXiv preprint arXiv:2407.07787*, 2024.

[11] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[12] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PmLR, 2020.

[13] Jorge Ramírez-Ruiz, Dmytro Grytskyy, Chiara Mastrogiuseppe, Yamen Habib, and Rubén Moreno-Bote. Complex behavior from intrinsic motivation to occupy future action-state path space. *Nature Communications*, 15(1):6368, 2024.

[14] Rubén Moreno-Bote and Jorge Ramirez-Ruiz. Empowerment, free energy principle and maximum occupancy principle compared. In *NeurIPS 2023 workshop: Information-Theoretic Principles in Cognitive Systems*, 2023.

[15] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

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

[17] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[18] Suguru Arimoto. An algorithm for computing the capacity of arbitrary discrete memoryless channels. *IEEE Transactions on Information Theory*, 18(1):14–20, 1972.

[19] Richard Blahut. Computation of channel capacity and rate-distortion functions. *IEEE transactions on Information Theory*, 18(4):460–473, 1972.

[20] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.

[21] Ming Xu, Matias Quiroz, Robert Kohn, and Scott A Sisson. Variance reduction properties of the reparameterization trick. In *The 22nd international conference on artificial intelligence and statistics*, pages 2711–2720. PMLR, 2019.

[22] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[23] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.

[24] Donald Wayne Bushaw. *Differential equations with a discontinuous forcing term*. Stevens Institute of Technology, 1953.

[25] Richard Bellman, Irving Glicksberg, and Oliver Gross. On the "bang-bang" control problem. *Quarterly of Applied Mathematics*, 14(1):11–18, 1956.

[26] Arash Tavakoli, Fabio Pardo, and Petar Kormushev. Action branching architectures for deep reinforcement learning. In *Proceedings of the aaai conference on artificial intelligence*, volume 32, 2018.

[27] OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

[28] Nino Vieillard, Marcin Andrychowicz, Anton Raichuk, Olivier Pietquin, and Matthieu Geist. Implicitly regularized rl with implicit q-values. *arXiv preprint arXiv:2108.07041*, 2021.

[29] Luke Metz, Julian Ibarz, Navdeep Jaitly, and James Davidson. Discrete sequential prediction of continuous actions for deep rl. *arXiv preprint arXiv:1705.05035*, 2017.

[30] Chen Tessler, Guy Tennenholtz, and Shie Mannor. Distributional policy optimization: An alternative approach for continuous control. *Advances in Neural Information Processing Systems*, 32, 2019.

[31] Andrey Sakryukin, Chedy Raïssi, and Mohan Kankanhalli. Inferring dqn structure for high-dimensional continuous control. In *International Conference on Machine Learning*, pages 8408–8416. PMLR, 2020.

[32] Arash Tavakoli, Mehdi Fatemi, and Petar Kormushev. Learning to represent action values as a hypergraph on the action vertices. *arXiv preprint arXiv:2010.14680*, 2020.

[33] Jonathan Chang, Masatoshi Uehara, Dhruv Sreenivas, Rahul Kidambi, and Wen Sun. Mitigating covariate shift in imitation learning via offline data with partial coverage. *Advances in Neural Information Processing Systems*, 34:965–979, 2021.

[34] Artemy Kolchinsky and Brendan D Tracey. Estimating mixture entropy with pairwise distances. *Entropy*, 19(7):361, 2017.

[35] Jiamin He, Samuel Neumann, Adam White, and Martha White. Investigating mixture policies in entropy-regularized actor-critic. 2025.

[36] Christian Daniel, Gerhard Neumann, Oliver Kroemer, and Jan Peters. Hierarchical relative entropy policy search. *Journal of Machine Learning Research*, 17(93):1–50, 2016.

[37] Iman Nematollahi, Erick Rosete-Beas, Adrian Röfer, Tim Welschehold, Abhinav Valada, and Wolfram Burgard. Robot skill adaptation via soft actor-critic gaussian mixture models. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 8651–8657. IEEE, 2022.

[38] Oleg Arenz, Mingjun Zhong, and Gerhard Neumann. Trust-region variational inference with gaussian mixture models. *Journal of Machine Learning Research*, 21(163):1–60, 2020.

[39] Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[40] Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *International conference on machine learning*, pages 1312–1320. PMLR, 2015.

[41] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. *arXiv preprint arXiv:1802.06070*, 2018.

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

[43] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.

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

[45] Seohong Park, Jongwook Choi, Jaekyeom Kim, Honglak Lee, and Gunhee Kim. Lipschitz-constrained unsupervised skill discovery. In *International Conference on Learning Representations (ICLR)*, 2022.

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

[47] Jacqueline Gottlieb, Pierre-Yves Oudeyer, Manuel Lopes, and Adrien Baranes. Information-seeking, curiosity, and attention: computational and neural mechanisms. *Trends in cognitive sciences*, 17(11):585–593, 2013.

[48] Joel Lehman and Kenneth O Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary computation*, 19(2):189–223, 2011.

[49] Alexander S Klyubin, Daniel Polani, and Chrystopher L Nehaniv. Empowerment: A universal agent-centric measure of control. In *2005 ieee congress on evolutionary computation*, volume 1, pages 128–135. IEEE, 2005.

[50] Felix Leibfried, Sergio Pascual-Diaz, and Jordi Grau-Moya. A unified bellman optimality principle combining reward maximization and empowerment. *Advances in Neural Information Processing Systems*, 32, 2019.

[51] Karl Friston, Philipp Schwartenbeck, Thomas FitzGerald, Michael Moutoussis, Timothy Behrens, and Raymond J Dolan. The anatomy of choice: active inference and agency. *Frontiers in human neuroscience*, 7:598, 2013.

[52] Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, et al. Gymnasium: A standard interface for reinforcement learning environments. *arXiv preprint arXiv:2407.17032*, 2024.

[53] Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Stable baselines. `https://github.com/hill-a/stable-baselines`, 2018.

[54] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[55] Boris T Polyak and Anatoli B Juditsky. Acceleration of stochastic approximation by averaging. *SIAM journal on control and optimization*, 30(4):838–855, 1992.

6 Appendix

6.1 Related Work

Discretization of Continuous Action Spaces: In control theory, the idea of discretizing continuous action spaces dates back to Bushaw's work on discontinuous forcing terms [24], and was formalized in the "bang–bang" control problem by [25]. However, naive uniform discretization suffers from an exponential explosion of possible actions in high-dimensional settings. To mitigate this, a family of methods assumes independence across action dimensions, discretizing each separately [26–28, 5]. Complementary approaches instead model inter-dimensional dependencies—either via sequential prediction of continuous actions [29], distributional policy optimization frameworks [30], learned Q-network structures for high-dimensional control [31], or hypergraph representations of action-value functions [32].

Offline and Demonstration-Based Action Quantization: Concurrently, offline and demonstration-based approaches have focused on deriving discrete action primitives directly from fixed data (not allowing the interaction of the agent with the environment). For example, state-conditioned action quantization has been implemented via VQ-VAE frameworks [6]. [7] proposed AQuaDem, which constructs a state-conditioned discretization of the continuous action space entirely offline using logged human demonstrations before any online reinforcement learning. Q-Transformer [8] adopts a per-dimension discretization that treats each action axis as a separate time step, predicting one bin at a time. This avoids the exponential grow of actions in multi-dimensional discretization, but it forces a coarse–fine trade-off in movement granularity. This important limitation has motivated contemporaneous work on quantization-free continuous sequence models such as the Q-FAT, which directly parameterize continuous actions via Gaussian mixture models, eliminating per-dimension binning [9]. However, like Q-Transformer's per-dimension discretization scheme, Q-FAT is trained entirely on offline demonstration datasets using a negative log-likelihood objective without access to reward signals, making it susceptible to distributional shift and dataset bias when deployed out-of-distribution [33]. Moreover, there is no guarantee that the mixture-model representation will maintain diverse component utilization (see e.g., [9]).

Mixture Policy Representations and Entropy Estimation: [34] introduce a family of analytic estimators for the differential entropy of mixture distributions, based on pairwise distance functions between components. [35] examines conditional Gaussian mixture policies within the entropy-regularized actor-critic framework, derives both half-reparameterization and Gumbel-softmax gradient estimators for the mixture weights and components, and demonstrates that mixture policies yield in some cases more robust exploration and higher solution quality than unimodal Gaussians across a variety of benchmarks. [36] extend the Relative Entropy Policy Search algorithm to hierarchical "mixed-option" policies by formulating policy learning as a latent-variable problem and incorporating an uncertainty constraint to enforce mode separation. Nematollahi et al. [37] propose a hybrid approach in which a Gaussian mixture dynamical system is first learned from demonstrations to model trajectory distributions of robot skills.

Static Variational–Inference vs. Bellman-Coupled RL: Both our method and VIPS [38] leverage log–responsibility updates to alternate between mixture components and weights, a structure rooted in Arimoto's algorithm. VIPS, however, is a *static* variational–inference procedure that maximizes an ELBO to approximate a fixed target distribution, whereas we optimize a *sequential* MDP objective in which the mixture–component entropy terms are embedded in a Bellman recursion, creating temporal coupling across states. Our objective also differs: we jointly *maximize* mixture entropy while *minimizing* per–component entropy (maxi–mix–mini–com), rather than performing a KL/ELBO projection. Finally, our tabular updates admit closed forms and we prove monotonic improvement of the value function per iteration, without trust–region or Monte Carlo approximated lower–bound steps.

Hierarchical RL: encompasses frameworks such as goal-conditioned RL, unsupervised skill discovery, and option-based methods that decompose tasks into high- and low-level policies [39]. Goal-conditioned approaches like Universal Value Function Approximators train policies to achieve explicit goals but require predefined goal signals [40], while skill-discovery methods (e.g., DIAYN [41], VALOR [42]) uncover diverse behaviors via mutual-information or empowerment objectives but often lack convergence guarantees or principled composition). Option-based frameworks learn temporally extended primitives under formal hierarchies but can depend on offline data or manual subgoal

design [43]. In contrast, our maxi-mix-mini-com framework discovers entropy-driven component policies fully online, providing provable convergence, explicit diversity control, and interpretable components that integrate seamlessly into hierarchical architectures.

Unsupervised Skill Discovery: Online unsupervised action quantization and unsupervised skill discovery differ fundamentally in their structure and deployment. Action quantization produces a set of atomic components that can be rapidly switched at every timestep, enabling fine-grained control and flexible composition within or across states. In contrast, skill discovery methods such as DIAYN, DADS, LSD, VALOR, and METRA learn temporally extended skills or options, each with its own intra-skill policy and often explicit termination conditions, so that switching typically occurs only at option boundaries [41, 44, 45, 42, 46]. The distinction targeted by each approach also diverges: action quantization explicitly encourages distinct actions in the same state by separating the supports of its components in action space, while skill discovery maximizes mutual information or related divergences between a latent skill variable and state, final state, or trajectory distributions, resulting in skills that may execute identical actions in the same state but diverge over extended rollouts. State dependence in action quantization arises from state-conditioned selection and the agent's controllability, without a direct objective for maximizing state-space coverage; skill discovery, in contrast, explicitly couples the latent variable to state visitation, thereby incentivizing broader exploration. In deployment, action quantization exposes a discrete set of selectable actions for standard discrete-action RL methods, requiring no additional termination handling, whereas skill discovery introduces a two-level control hierarchy that manages skill selection, intra-skill policy execution, and option termination. As a result, compositional control is fine-grained and per-step with action quantization, but is inherently coarser and limited to skill boundaries in skill discovery frameworks. Our preliminary experiments (not shown) indicate, as expected, that skill discovery approaches used as quantized action provide lower performance than our maxi-mix-mini-com approach.

Intrinsic Motivation addresses the question of what reward-agnostic signals can generate behavior useful for learning and discovery. Several frameworks have emerged where intrinsic signals are major drives of behavior: information seeking [47], novelty seeking [48], empowerment [49, 50], minimizing free energy [51], or occupying action-state space [13]. We have focused on the latter because the maximum occupancy principle (MOP) provides a principled way to generate diverse, complex behaviors only limited by the constraints of the agent and the presence of terminal states. Empowerment maximizes the mutual entropy between future states and sequences of actions, and therefore it is an appealing objective to generate diverse behavior. However, it has been shown that empowerment provides no clear closed-form recursion beyond its per-step formulation [14]. In any event, it is worthy exploring in future work different intrinsic motivation objectives for action-policy discretization.

6.2 Monotonic Policy Iteration for the Maxi-Mix-Mini-Comp Entropy Objective

Here we provide a full proof of Theorems 1 and 2. We reproduce some results previously shown in Sec. 2 to ease understanding. First we remind the reader that our objective can be expressed as maximizing the value function over component and coefficients as the optimality Bellman equation

$$V^*(s) = \max_{\pi,w} \left(R(s;\pi,w) + \gamma \sum_a \pi_m(a|s) Q^*(s,a) \right) \tag{18}$$

$$R(s;\pi,w) \equiv \mathcal{H}(\pi_m(\cdot|s)) - \alpha \sum_k w_k(s) \mathcal{H}(\pi_k(\cdot|s)) \,, \tag{19}$$

where π indicates the set of component policies $\pi = \{\pi_1, ..., \pi_K\}$ and $w = \{w_1, ..., w_K\}$, where the conditioning on the state s is omitted but understood. Note that the sum limits in $\sum_a$ are state-dependent, as we in general allow for state-dependent action set $\mathcal{A}(s)$.

If $\alpha < 1$, the immediate reward is non-negative, $R(s;\pi,w) \geq 0$, because the entropy of a mixture is always larger or equal than its maximum entropy component – easy to prove using concavity of $-x \log x$ in $[0, 1]$. Since the immediate reward $R(s;\pi,w)$ is bounded for all s in finite action spaces, it is not difficult to show that the optimal value function V^* admits an unique solution, although possibly through non-unique optimal π and w (see e.g., steps in [15], using the fact that if $0 \leq \gamma < 1$, then the optimal Bellman operator is a contraction).

We first prove that for a high capacity mixture policy, that is, when the number of components is larger or equal than the action space, then solving Eq. 18 is equivalent to solving an unconstrained problem with no restrictions on the mixture policy.

Theorem 1. *If $K(s) \geq |\mathcal{A}(s)|$ for all s, then the optimal solution of Eq. 18 is such that $\pi_k(a_k|s) = 1$ only for one action a_k and the set of actions spans $\mathcal{A}(s)$, that is, $\{a_k\} = \mathcal{A}(s)$. Then, π_m in Eq. 1 is unconstrained and the optimal value function obeys*

$$V^*(s) = \max_{\pi,w} \left(\mathcal{H}(\pi_m(\cdot|s_t)) + \gamma \sum_a \pi_m(a|s) Q^*(s,a) \right) \tag{20}$$

The optimal mixture policy is unique, while the mixture components are not unique if $|\mathcal{A}(s)| > 1$ at least for one state.

Proof. It is obvious that for any fixed mixture policy π_m, the value function in Eq. 3 is maximized if each of the components can be made to have zero entropy (as $\alpha > 0$), eliminating the negative contribution to the immediate reward. Zero entropy components can be obtained iff for each component k, there exists a single action a_k such that $\pi_k(a_k|s) = 1$. As $K(s) \geq |\mathcal{A}(s)|$, and defining $\mathcal{A}(s) = \{a_1, ..., a_{\mathcal{A}(s)}\}$, we can choose the components π_k such that $\pi_k(a_k|s) = 1$ (that is, it is a deterministic component policy) for $k = 1, ..., |\mathcal{A}|(s)$, and $\pi_k(a_1|s) = 1$ for $k > |\mathcal{A}|(s)$. Then, any policy $\pi(a|s)$ can be built from these components using $w_k(s) = \pi(a_k|s)$ in Eq. 1 for $k \leq |\mathcal{A}(s)|$ and $w_k(s) = 0$ otherwise. This means that the mixture policy is unconstrained, and then it can be made to optimize the Bellman Eq. 5 with $\alpha = 0$, or equivalently Eq. 20. Non uniqueness of the components is clear because we are free to permute assignments between actions and components. Uniqueness of V^* and the optimal policy π^*_m has been shown in [13]. □

For the case $K(s) < |\mathcal{A}(s)|$, we can write the optimality Bellman equation 18 (see Sec. 2) as

$$V^*(s) = \max_{\pi,w} \max_r \left(G(s;\pi,w,r) + \gamma \sum_a \pi_m(a|s) Q^*(s,a) \right) \tag{21}$$

Our algorithm (see Sec. 2) proceeds by following the next steps:

(1) Initial conditions: Start the value function $V^{(0)}(s) = 0$ for all s and arbitrary $\pi^{(0)} = \{\pi_1^{(0)}, ..., \pi_K^{(0)}\}$ and $w^{(0)} = \{w_1^{(0)}, ..., w_K^{(0)}\}$ outside the simplex boundaries, that is, $\pi_k^{(0)}(a|s) > 0$ and $w_k^{(0)}(s) > 0$ for all k, s and a. Define the initial responsibilities as

$$r_k^{(0)}(s,a) = \frac{w_k^{(0)}(s)\, \pi_k^{(0)}(a|s)}{\pi_m^{(0)}(a|s)} > 0 \,. \tag{22}$$

Iterate the following step (2) for $n = 1, 2, ...$

(2) At iteration n, start from $V^{(n-1)}$, $\pi^{(n-1)}$, $w^{(n-1)}$ and $r^{(n-1)}$. Perform the following steps, (2a) to (2d).

(2a) Define the updated value function $V^{(n)}$ as

$$V^{(n)}(s) = G(s;\pi^{(n-1)}, w^{(n-1)}, r^{(n-1)}) + \gamma \sum_a \pi_m^{(n-1)}(a|s) Q^{(n-1)}(s,a) \,, \tag{23}$$

with $Q^{(n-1)}(s,a) = \sum_{s'} p(s'|s,a) V^{(n-1)}(s)$ and $\pi_m^{(n-1)} = \sum_k w_k^{(n-1)}(s) \pi_k^{(n-1)}(a|s)$.

Note that with this definition, we have $V^{(1)}(s) \geq V^{(0)}(s) = 0$ for all s, because the gain $G(s;\pi^{(0)}, w^{(0)}, r^{(0)}) = R(s;\pi^{(0)}, w^{(0)})$ is non-negative.

(2b) Compute $\pi^{(n)}$ fixing $V^{(n-1)}$ and $r^{(n-1)}$ using

$$\pi_k^{(n)}(a|s) = \frac{[r_k^{(n-1)}(s,a)]^{\frac{1}{1-\alpha}} e^{\frac{\gamma}{1-\alpha} Q^{(n-1)}(s,a)}}{\sum_{a'} [r_k^{(n-1)}(s,a')]^{\frac{1}{1-\alpha}} e^{\frac{\gamma}{1-\alpha} Q^{(n-1)}(s,a')}} \,. \tag{24}$$

This equation is the result of the optimization

$$\pi^{(n)}(\cdot|s) = \arg\max_{\pi} \left(G(s;\pi, w^{(n-1)}, r^{(n-1)}) + \gamma \sum_{k,a} w_k^{(n-1)}(s)\, \pi_k(a|s) Q^{(n-1)}(s,a) \right) , \quad (25)$$

where $\pi^{(n)}(\cdot|s) = \{\pi_1^{(n)}(\cdot|s), \cdots, \pi_K^{(n)}(\cdot|s)\}$, as it can be checked adding Lagrange multipliers and taking derivatives (see details in Appendix 6.2.1). Note that the ordering of the component policies $\pi_1, \ldots, \pi_K$ is immaterial, as the objective in Eq. 25 is additive over k, and each π_k can be optimized independently when $w^{(n-1)}$ and $r^{(n-1)}$ are fixed. Consequently, the solution is invariant to any permutation of the indices. The solution $\pi_k^{(n)}(a|s)$ provided in Eq. 24 is the only global optima if $w_k^{(n-1)}(s) > 0$ for all k and s, as then the solution lies inside the simplex and the objective is strictly concave.

Defining $V_\pi^{(n)}$ as the argument of the right hand side of Eq. 25, after replacing π with the optimal $\pi_k^{(n)}(a|s)$, we obtain

$$V_\pi^{(n)} = G(s;\pi^{(n)}, w^{(n-1)}, r^{(n-1)}) + \gamma \sum_{k,a} w_k^{(n-1)}(s)\, \pi_k^{(n)}(a|s) Q^{(n-1)}(s,a) \quad (26)$$

$$= (1-\alpha) \sum_k w_k^{(n-1)}(s) \log \left[\sum_a [r_k^{(n-1)}(s,a)]^{\frac{1}{1-\alpha}} e^{\frac{\gamma}{1-\alpha} Q^{(n-1)}(s,a)} \right]$$
$$- \sum_k w_k^{(n-1)}(s) \log(w_k^{(n-1)}(s)) . \quad (27)$$

Naturally, we have $V_\pi^{(n)}(s) \geq V^{(n)}(s)$ for all s, because after the optimization the quantity in the bracket on right hand side of Eq. 25 cannot decrease.

(2c) Compute $w^{(n)}$ fixing $V^{(n-1)}$ and $r^{(n-1)}$ using

$$w_k^{(n)}(s) = \frac{\left(\sum_a [r_k^{(n-1)}(s,a)]^{\frac{1}{1-\alpha}} e^{\frac{\gamma}{1-\alpha} Q^{(n-1)}(s,a)} \right)^{1-\alpha}}{\sum_l \left(\sum_a [r_l^{(n-1)}(s,a)]^{\frac{1}{1-\alpha}} e^{\frac{\gamma}{1-\alpha} Q^{(n-1)}(s,a)} \right)^{1-\alpha}} . \quad (28)$$

This equation is the result of the optimization

$$w^{(n)}(s) = \arg\max_{w} \left(G(s;\pi^{(n)}, w, r^{(n-1)}) + \gamma \sum_{k,a} w_k(s)\, \pi_k^{(n)}(a|s) Q^{(n-1)}(s,a) \right) , \quad (29)$$

by using the previously computed $\pi^{(n)}$ in step (2b), and where we denote $w^{(n)}(s) = \{w_1^{(n)}(s), \cdots, w_K^{(n)}(s)\}$. The solution in Eq. 28 is the only global maximum provided that $r_k^{(n-1)}(s,a) > 0$ for all k, s and a (see Appendix 6.2.3 for details).

Defining $V_{\pi,w}^{(n)}$ as the argument of the right hand side of Eq. 29, after replacing w with the optimal $w_k^{(n)}(s)$, we obtain

$$V_{\pi,w}^{(n)} = G(s;\pi^{(n)}, w^{(n)}, r^{(n-1)}) + \gamma \sum_{k,a} w_k^{(n)}(s)\, \pi_k^{(n)}(a|s) Q^{(n-1)}(s,a) \quad (30)$$

$$= \log \left[\sum_k \left(\sum_a [r_k^{(n-1)}(s,a)]^{\frac{1}{1-\alpha}} e^{\frac{\gamma}{1-\alpha} Q^{(n-1)}(s,a)} \right)^{1-\alpha} \right] . \quad (31)$$

Naturally, we have $V_{\pi,w}^{(n)}(s) \geq V_\pi^{(n)}(s)$ for all s, because after the optimization the right hand side of Eq. 29 cannot decrease. This, in turn, implies that $V_{\pi,w}^{(n)}(s) \geq V^{(n)}(s)$ using the last inequality in step (2a).

(2d) Compute $r^{(n)}$ fixing $V^{(n-1)}$, $\pi^{(n)}$ and $w^{(n)}$ using

$$r_k^{(n)}(s,a) = \frac{w_k^{(n)}(s)\, \pi_k^{(n)}(a|s)}{\pi_m^{(n)}(a|s)} , \quad (32)$$

with $\pi_m^{(n)}(a|s) = \sum_k w_k^{(n)}(s)\, \pi_k^{(n)}(a|s)$. This equation is the result of the optimization (see Appendix 6.2.3 for details),

$$r^{(n)}(s,\cdot) = \arg\max_r \left(G(s;\pi^{(n)}, w^{(n)}, r) + \gamma \sum_{k,a} w_k^{(n)}(s)\, \pi_k^{(n)}(a|s) Q^{(n-1)}(s,a) \right) , \tag{33}$$

where $r^{(n)}(s,\cdot) = \{r_1^{(n)}(s,\cdot), \cdots, r_K^{(n)}(s,\cdot)\}$.

Defining $V_{\pi,w,r}^{(n)}$ as the quantity within the bracket on the right hand side of Eq. 33 after replacing r with the optimal $r_k^{(n)}(s,a)$, we obtain

$$V_{\pi,w,r}^{(n)} = G(s;\pi^{(n)}, w^{(n)}, r^{(n)}) + \gamma \sum_{k,a} w_k^{(n)}(s)\, \pi_k^{(n)}(a|s) Q^{(n-1)}(s,a) \, . \tag{34}$$

Naturally, we have $V_{\pi,w,r}^{(n)}(s) \geq V_{\pi,w}^{(n)}(s)$ for all s, because after the optimization the right hand side of Eq. 33 cannot decrease. This, in turn, implies that $V_{\pi,w,r}^{(n)}(s) \geq V^{(n)}(s)$ using steps (2a) and (2b).

Theorem 2. *The algorithm following steps (1) and (2), which consists in iterating the π, w and r using Eqs. 23,24,28,32 in this order, improves the value function monotonically until convergence.*

Proof. From (2a) above we have seen that $V^{(1)} \geq V^{(0)}$. Assume that it is true $V^{(n)} \geq V^{(n-1)}$ for some $n > 0$. Then, from the definition of V^{n+1} using Eq. 23 we see that

$$\begin{aligned} V^{(n+1)}(s) &= G(s;\pi^{(n)}, w^{(n)}, r^{(n)}) + \gamma \sum_a \pi_m^{(n)}(a|s) Q^{(n)}(s,a) && (35)\\ &\geq G(s;\pi^{(n)}, w^{(n)}, r^{(n)}) + \gamma \sum_a \pi_m^{(n)}(a|s) Q^{(n-1)}(s,a) && (36)\\ &= V_{\pi,w,r}^{(n)} && (37)\\ &\geq V^{(n)}(s) \, , && (38) \end{aligned}$$

where in the second line we have used the induction assumption, implying that $Q^{(n)}(s,a) \geq Q^{(n-1)}(s,a)$ for all (s,a), in the third line we have used the definition of $V_{\pi,w,r}^{(n)}$ in Eq. 34, and in the last line we have used the final implication in (2d).

Then, by induction, the series $V^{(n)}(s)$ is non-decreasing for all s, and since the value function is finite ($V(s) \leq \log(\max_s |\mathcal{A}(s)|)/(1-\gamma)$ for any policy π_m), the series converges to a finite value $V^{(\infty)}(s)$ as n goes to infinity. There is strick improvement ($>$ instead of $\geq$ in all equations above) whenever the updated π, w or r are different from the previous ones.

We finally check for consistency of Eqs. 24,28,32, as these optima are obtained by taking derivatives with Lagrange multipliers and assuming that the optima do not lie on the simplex boundaries. The initial condition indeed obeys this, $\pi_k^{(0)}(a|s) > 0$, $w_k^{(0)}(s) > 0$ and $r_k^{(0)}(s,a) > 0$ for all k, s and a. Then, positivity of all variables is true for $n = 0$. Assuming that positivity of all variables is true for some n such that $n > 0$, then it can be shown that it is again true for $n+1$ using Eqs. 24,28,32. By induction, it follows that the optima are always in the simplex but outside the boundary, although they can get infinitely close to the simplex boundary as n goes to infinity, meaning that the converged solution might actually be at the boundary. □

Remark: The limit $V^{(\infty)}(s)$ must be a solution to the Bellman Eq. 3. Therefore, it corresponds to a proper value function with a proper policy. Indeed, the policy must converge as well to a policy $\pi_m^{(\infty)}(a|s)$, because it cannot change when the value function $V^{(n)}(s)$ has converged: if the policy changed, then there should an strict ($>$) improvement of the value function in every new iteration, which contradicts the assumption that the value function has converged. However, convergence of the series $V^{(n)}(s)$ does not imply that the optimal Bellman equation is solved by $V^{(\infty)}(s)$, so it might lead to a suboptimal value and policy. The converged values also might depend on the initial conditions.

Remark: Any orderings of the updates for π, w or r lead to equivalent algorithms. This is because π and w only depend on r and Q, so their order is interchangable, and any other update ordering is just a cyclic permutation of one of these two.

6.2.1 Finding optimal component policies at iteration n

At iteration n, with fixed value function $V^{(n-1)}$, mixture weights $w^{(n-1)}$ and responsibilities $r^{(n-1)}$, the update for the component policies π_k is given by the maximization problem

$$\pi^{(n)}(\cdot|s) = \arg\max_{\pi} \left(G(s; \pi, w^{(n-1)}, r^{(n-1)}) + \gamma \sum_{k,a} w_k^{(n-1)}(s)\, \pi_k(a|s) Q^{(n-1)}(s,a) \right).$$

In this optimization, for a given k, only the following terms depending on π_k contribute,

$$-(1-\alpha) \sum_a \pi_k(a|s) \log \pi_k(a|s) + \sum_a \pi_k(a|s) \left[\log r_k^{(n-1)}(s,a) + \gamma Q^{(n-1)}(s,a) \right],$$

subject to $\sum_a \pi_k(a \mid s) = 1$. This expression is multiplied by $w_k^{(n-1)}(s)$, but because $w_k^{(n-1)}(s) > 0$ for all k, s and n (see Theorem 2), it can be removed. We note that for $\alpha < 1$ this expression is strictly concave in $\pi_k(\cdot|s)$; therefore if there is a critical point in the simplex $\sum_a \pi_k(a \mid s) = 1$, it will be the global maximum, which will be the case as shown below.

We note that the simplification that arises from having a constant weight $w_k^{(n-1)}(s)$ multiplying the previous equation does not hold if the weighted sum of entropies in Eq. 2 is replaced by an unweighted sum (i.e. replacing the weighted entropy penalty $\sum_k w_k(s)\, \mathcal{H}(\pi_k(\cdot \mid s))$ in Eq. (2) with the *unweighted* sum $\sum_k \mathcal{H}(\pi_k(\cdot \mid s))$). In this case, the first term (multiplied by the weight) would be replaced by $-(w_k^{(n-1)}(s) - \alpha) \sum_a \pi_k(a|s) \log \pi_k(a|s)$, leading to a non-concave problem depending on the current sign of $-(w_k^{(n-1)}(s) - \alpha)$ at iteration $n-1$, which could also fluctuate across iterations (remember that $\alpha < 1$ and $0 < w_k^{(n-1)}(s) < 1$).

Introducing a Lagrange multiplier λ_k, we form the Lagrangian

$$\begin{aligned}\mathcal{L}_k(s, \pi_k, \lambda_k) = &\sum_a \pi_k(a|s) \Big[\log r_k^{(n-1)}(s,a) + \gamma Q^{(n-1)}(s,a) \Big] - (1-\alpha) \sum_a \pi_k(a|s) \log \pi_k(a|s) \\ &+ \lambda_k \left(\sum_a \pi_k(a|s) - 1 \right).\end{aligned}$$

Differentiating with respect to $\pi_k(a|s)$ and setting it to zero to find critical points gives

$$\frac{\partial \mathcal{L}_k}{\partial \pi_k(a|s)} = \log r_k^{(n-1)}(s,a) + \gamma Q^{(n-1)}(s,a) - (1-\alpha) \Big(\log \pi_k(a|s) + 1 \Big) + \lambda_k = 0, \tag{39}$$

which after rearranging becomes

$$\log \pi_k(a|s) = \frac{\log r_k^{(n-1)}(s,a) + \gamma Q^{(n-1)}(s,a) + \lambda_k - (1-\alpha)}{1-\alpha},$$

or equivalently,

$$\pi_k(a|s) = \exp\left(\frac{\log r_k^{(n-1)}(s,a) + \gamma Q^{(n-1)}(s,a)}{1-\alpha} \right) \exp\left(\frac{\lambda_k - (1-\alpha)}{1-\alpha} \right).$$

Enforcing normalization yields the closed-form softmax update

$$\pi_k^{(n)}(a|s) = \frac{[r_k^{(n-1)}(s,a)]^{\frac{1}{1-\alpha}}\, e^{\frac{\gamma}{1-\alpha} Q^{(n-1)}(s,a)}}{\sum_{a'} [r_k^{(n-1)}(s,a')]^{\frac{1}{1-\alpha}}\, e^{\frac{\gamma}{1-\alpha} Q^{(n-1)}(s,a')}}.$$

identical to Eq. 14.

Because $r_k^{(n-1)}(s,a) > 0$ and $Q^{(n-1)}(s,a) < \infty$, the critical point is in the simplex. Therefore $w_k^{(n)}$ is the global maximum. Further, note that the global maximum is not on the simplex boundaries. We remind the reader that the fact that $\pi^{(n)}, w^{(n)}, r^{(n)}$ are all positive for all n has been shown by induction (see Theorem 2).

6.2.2 Finding optimal mixture weights at iteration n

To determine the optimal weights $w_k^{(n)}(s)$, we solve the constrained maximization

$$w_k^{(n)}(s) = \arg\max_{w} \Big[G\big(s;\, \pi^{(n)}, w, r^{(n-1)}\big) + \gamma \sum_{k,a} w_k(s)\, \pi_k^{(n)}(a \mid s)\, Q^{(n-1)}(s,a) \Big], \quad \text{subject to} \quad \sum_k w_k(s) = 1, \tag{40}$$

where $\pi^{(n)}$ is computed in step (2b) and $r_k^{(n-1)}(s,a) > 0$ for all k, s, a.

We can rewrite the objective as we saw earlier (see Eq. 27) as

$$\begin{aligned} &G\big(s;\, \pi^{(n)}, w, r^{(n-1)}\big) + \gamma \sum_{k,a} w_k(s)\, \pi_k^{(n)}(a \mid s)\, Q^{(n-1)}(s,a) \\ &= (1-\alpha) \sum_k w_k(s) \log F_k^{(n-1)}(s) - \sum_k w_k(s) \log w_k(s)\ , \end{aligned} \tag{41}$$

where

$$F_k^{(n-1)}(s) = \sum_a [r_k^{(n-1)}(s,a)]^{\frac{1}{1-\alpha}}\, e^{\frac{\gamma}{1-\alpha}\ Q^{(n-1)}(s,a)}\ . \tag{42}$$

We note again that the objective is strictly concave in the $w_k(s)$, and therefore once again if there is a critical point in the simplex $\sum_k w_k(s) = 1$, it must be the global maximum, as it is shown below.

To enforce the simplex constraint, we introduce a Lagrange multiplier λ_w and form

$$\mathcal{L}\big(s, w, \lambda_w\big) = (1-\alpha) \sum_{k=1}^{K} w_k(s)\ \log F_k^{(n-1)}(s) - \sum_{k=1}^{K} w_k(s)\ \log w_k(s) + \lambda_w \Big(\sum_{k=1}^{K} w_k(s) - 1 \Big)\,. \tag{43}$$

Differentiating $\mathcal{L}$ with respect to $w_k(s)$ and setting the derivative to zero yields

$$-\log w_k(s) - 1 + (1-\alpha)\ \log F_k^{(n-1)}(s) + \lambda_w = 0. \tag{44}$$

Solving for $w_k(s)$ and enforcing $\sum_k w_k(s) = 1$ gives the closed-form update

$$w_k^{(n)}(s) = \frac{\big(F_k^{(n-1)}(s)\big)^{1-\alpha}}{\sum_l \big(F_l^{(n-1)}(s)\big)^{1-\alpha}} = \frac{\Big(\sum_a [r_k^{(n-1)}(s,a)]^{\frac{1}{1-\alpha}}\, e^{\frac{\gamma}{1-\alpha} Q^{(n-1)}(s,a)}\Big)^{1-\alpha}}{\sum_l \Big(\sum_a [r_l^{(n-1)}(s,a)]^{\frac{1}{1-\alpha}}\, e^{\frac{\gamma}{1-\alpha}\ Q^{(n-1)}(s,a)}\Big)^{1-\alpha}}\,. \tag{45}$$

Because $r_k^{(n-1)}(s,a) > 0$ and $Q^{(n-1)}(s,a) < \infty$, the optimal mixture weights at iteration n are in its simplex, but not on its simplex boundaries.

6.2.3 Finding optimal responsibilities at iteration n

To solve the optimization problem

$$r^{(n)}(s,\cdot) = \arg\max_{r} \left(G(s; \pi^{(n)}, w^{(n)}, r) + \gamma \sum_{k,a} w_k^{(n)}(s)\, \pi_k^{(n)}(a|s) Q^{(n-1)}(s,a) \right)\ ,$$

we note that the only term involving r affecting the maximization is

$$\sum_{k,a} w_k^{(n)}(s)\, \pi_k^{(n)}(a|s)\ \log r_k(s,a),$$

subject to the normalization constraint that for every state-action pair (s,a) the responsibilities satisfy $\sum_{k=1} r_k(s,a) = 1$. Note that when $r_k^{(n)}(s,a) > 0$ and $\pi_k^{(n)}(a|s) > 0$ the function is strictly concave in r. Therefore, if there is a critical point inside the simplex $\sum_k r_k(s,a) = 1$, it must be the global maximum, as we show below.

Introducing a Lagrange multiplier $\lambda_r(a|s)$ for each state–action pair (s, a), the Lagrangian can be written as

$$\mathcal{L}_r = \sum_{k=1}^{K} w_k^{(n)}(s)\, \pi_k^{(n)}(a|s)\, \log r_k(a|s) + \lambda_r(s,a) \left(1 - \sum_{k=1}^{K} r_k(s,a)\right). \tag{46}$$

Taking the derivative of the Lagrangian in (46) with respect to $r_k(a|s)$ yields

$$\frac{\partial \mathcal{L}_r}{\partial r_k(a|s)} = \frac{w_k^{(n)}(s)\, \pi_k^{(n)}(a|s)}{r_k(a|s)} - \lambda_r(a|s) = 0,$$

which implies

$$r_k(s,a) = \frac{w_k^{(n)}(s)\, \pi_k^{(n)}(a|s)}{\lambda_r(a|s)}.$$

Enforcing the normalization constraint

$$\sum_{k=1}^{K} r_k(a|s) = 1 \quad \Longrightarrow \quad \lambda_r(a|s) = \sum_{k=1}^{K} w_k^{(n)}(s)\, \pi_k^{(n)}(a|s) = \pi_m^{(n)}(a|s),$$

we obtain the updated responsibilities

$$r_k^{(n)}(a|s) = \frac{w_k^{(n)}(s)\, \pi_k^{(n)}(a|s)}{\pi_m^{(n)}(a|s)}\,.$$

These probabilities over k are the global maximum, and they do not lie on the simplex boundaries because $r_k^{(n)} > 0$ and $\pi_k^{(n)}(a|s) > 0$ (see Theorem 2).

6.3 Unsupervised Policy Gradient Reparameterization for Continuous Control

In this section, we present the proof of Theorem 3, in which the performance measure is defined as the value of the initial state of the episode, $J(\theta) = V(s_0)$. However, the derivation generalizes straightforwardly to the case where the initial state is drawn from a distribution $\rho(s_0)$, in which case the objective becomes $J(\theta) = \mathbb{E}_{s_0 \sim \rho}[V(s_0)]$, and the gradient is taken with respect to this expectation. Our derivation is a simple generalization of the policy gradient theorem of [39] by considering a mixture policy and incorporating, as intrinsic reward, a linear combination of the entropy of the mixture and the entropies of its component policies.

Theorem 3. *Let $\pi_{m,\theta}(a|s)$ be a mixture policy over continuous actions, and define the state-value function as in Eq. 3. Then the gradient of the performance objective $J(\theta) = V(s_0)$ admits the form*

$$\nabla_\theta J(\theta) = \int_{s,a} d_{\pi_{m,\theta}}(s) \pi_{m,\theta}(a|s) \Big(\nabla_\theta \log \pi_{m,\theta}(a|s) Q(s,a) + \nabla_\theta R(s; \pi_\theta, w_\theta) \Big)\, da\, ds.$$

where $\nabla_\theta R(s; \pi_\theta, w_\theta) = \nabla_\theta \mathcal{H}\big(\pi_{m,\theta}(\cdot|s)\big) - \alpha \nabla_\theta \sum_{k=1}^{K} w_{k,\theta}(s)\, \mathcal{H}(\pi_{k,\theta}(\cdot|s))$, $\pi_\theta = \{\pi_{1,\theta}, ..., \pi_{K,\theta}\}$ *and* $w_\theta = \{w_{1,\theta}, ..., w_{K,\theta}\}$

Proof. The gradient of the state-value function can be written in terms of the action-value function as

$$
\begin{aligned}
&\nabla_\theta V(s) \\
&= \nabla_\theta R(s;\pi_\theta,w_\theta) + \nabla_\theta \gamma \int_a \pi_{m,\theta}(a|s) Q(s,a)\, da \\
&= \nabla_\theta R(s;\pi_\theta,w_\theta) + \gamma \int_a \nabla_\theta \pi_{m,\theta}(a|s) Q(s,a)\, da + \gamma \int_a \pi_{m,\theta}(a|s) \nabla_\theta Q(s,a)\, da \\
&= \nabla_\theta R(s;\pi_\theta,w_\theta) + \gamma \int_a \nabla_\theta \pi_{m,\theta}(a|s) Q(s,a)\, da + \gamma \int_a \pi_{m,\theta}(a|s) \nabla_\theta \int_{s'} p(s'|s,a) V(s')\, ds\, da \\
&= \nabla_\theta R(s;\pi_\theta,w_\theta) + \gamma \int_a \nabla_\theta \pi_{m,\theta}(a|s) Q(s,a)\, da + \gamma \int_{a,s'} \pi_{m,\theta}(a|s) p(s'|s,a) \nabla_\theta V(s')\, ds\, da \\
&= \nabla_\theta R(s;\pi_\theta,w_\theta) + \gamma \int_a \nabla_\theta \pi_{m,\theta}(a|s) Q(s,a)\, da + \\
&+ \gamma \int_{a,s'} \pi_{m,\theta}(a|s) p(s'|s,a) \\
&\left(\nabla_\theta R(s';\pi_\theta,w_\theta) + \int_{a'} \nabla_\theta \pi_{m,\theta}(a'|s') Q(s',a')\, da' + \gamma \int_{a',s''} \pi_{m,\theta}(a'|s') p(s''|s',a') \nabla_\theta V(s'')\, ds'\, da' \right) \\
&= \int_x \sum_t^\infty Pr(s \to x, t, \pi_{m,\theta}) \left(\nabla_\theta R(s;\pi_\theta,w_\theta) + \int_a \nabla_\theta \pi_{m,\theta}(a|x) Q(x,a) \right)
\end{aligned}
\tag{47}
$$

where we define $Pr(s \to x, t, \pi_\theta)$ similar to [39] as the discounted probability of transitioning from state s to state x in t steps under the mixture policy $\pi_{m,\theta}$. Then we can find that

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta V(s_0) \\
&= \int_s \sum_t^\infty Pr(s_0 \to s, t, \pi_{m,\theta}) \left(\nabla_\theta R(s;\pi_\theta,w_\theta) + \int_a \nabla_\theta \pi_{m,\theta}(a|s) Q(s,a) \right) \\
&= \int_{s,a} d_{\pi_{m,\theta}}(s) \pi_{m,\theta}(a|s) \left(\nabla_\theta \log \pi_{m,\theta}(a|s) Q(s,a) + \nabla_\theta R(s;\pi_\theta,w_\theta) \right)
\end{aligned}
\tag{48}
$$

□

In this work, we reparametrize our component policies as $\pi_{k,\theta}(a|s) = p(\varepsilon)$ where $a = f_\theta(\varepsilon; s, k)$ and $p(\cdot)$ is the noise distribution. Here we prove theorem 4 (see Sec. 3) in which we provide an unbiased reparametrization of our component policies.

Theorem 4. *Let $\hat{f}_\theta = f_\theta(\varepsilon; s, k)$ then the gradients of objective function under our reparametrization become*

$$
\begin{aligned}
&\nabla_\theta J\big(\pi_{m,\theta}\big) \\
&= \mathbb{E}_{\substack{s\sim d_{\pi_{m,\theta}} \\ k\sim Cat(w_{k,\theta}(s)) \\ \varepsilon\sim p}} \Bigg[\nabla_\theta \log w_{k,\theta}(s) \left(Q_{\pi_{m,\theta}}\big(s, \hat{f}_\theta\big) - \log \pi_{m,\theta}\big(\hat{f}_\theta|s\big) + \alpha \log \pi_{k,\theta}\big(\hat{f}_\theta|s\big) \right) \\
&\qquad + \ \nabla_\theta \hat{f}_\theta \ \nabla_a \Big(Q_{\pi_{m,\theta}}(s,a) - \log \pi_{m,\theta}(a|s) + \alpha \log \pi_{k,\theta}(a|s) \Big) \Big|_{a=\hat{f}_\theta} \Bigg],
\end{aligned}
\tag{49}
$$

where $d_{\pi_{m,\theta}}(s)$ is the discounted occupancy measure under $\pi_{m,\theta}$.

Proof. Our proof depends heavily on similar proof provided by [35], where they discuss the reparametrization trick for entropy regularized policy gradient theorem for mixture policies. The only difference between our case and their case is the weighted sum of component entropy and the absence of the extrensic reward from the environment. We start with the extended policy gradient Theorem 3 which states that

$$
\nabla_\theta J(\theta) = \int_{s,a} d_{\pi_{m,\theta}}(s) \pi_{m,\theta}(a|s) \left(\nabla_\theta \log \pi_{m,\theta}(a|s) Q(s,a) + \nabla_\theta R(s;\pi_\theta,w_\theta) \right) da\, ds\,. \tag{50}
$$

We can break this last equation into 3 parts that we can handle separately. The first part is for the state-action value function which recover the vanilla policy gradient theorem [39]. [35] found that

$$\begin{aligned}&\int_{s,a} d_{\pi_{m,\theta}}(s)\pi_{m,\theta}(a|s)\nabla_\theta \log \pi_{m,\theta}(a|s) Q_{\pi_{m,\theta}}(s,a)\, da\, ds \\ &= \mathbb{E}_{\substack{s\sim d_{\pi_{m,\theta}} \\ k\sim \text{Cat}(w_{k,\theta}(s)) \\ \epsilon\sim p(\epsilon)}} \Big[Q_{\pi_{m,\theta}}\big(s,\, \hat{f}_\theta\big)\, \nabla_\theta \log w_{k,\theta}(s) + \nabla_\theta \hat{f}_\theta\, \nabla_a Q_{\pi_{m,\theta}}(s,a)\big|_{a=\hat{f}_\theta} \Big].\end{aligned} \tag{51}$$

The second part handle the entropy of the mixture policy and similarly [35] found that:

$$\int_{s,a} d_{\pi_{m,\theta}}(s)\, \pi_{m,\theta}(a \mid s)\, \nabla_\theta \mathcal{H}\big(\pi_{m,\theta}(\cdot \mid s)\big)\, da\, ds \tag{52}$$

$$= -\mathbb{E}_{\substack{s\sim d_{\pi_{m,\theta}} \\ k\sim \text{Cat}(w_{k,\theta}(s)) \\ \epsilon\sim p(\epsilon)}} \Big[\log \pi_{m,\theta}\big(\hat{f}_\theta \mid s\big)\, \nabla_\theta \log w_{k,\theta}(s) + \nabla_\theta \hat{f}_\theta\, \nabla_a \log \pi_{m,\theta}(a \mid s)\big|_{a=\hat{f}_\theta} \Big]. \tag{53}$$

The third term is

$$\int_{s,a} d_{\pi_{m,\theta}}(s)\pi_\theta(a|s)\nabla_\theta \left(\sum_k w_{k,\theta}(s)\mathcal{H}(\pi_{k,\theta}(\cdot|s)) \right) da\, ds\,. \tag{54}$$

Note first that we can rewrite the entropy of component k with reparametrization as:

$$\mathcal{H}(\pi_{k,\theta}(\cdot \mid s)) = -\,\mathbb{E}_{\epsilon\sim p(\epsilon)} \Big[\log \pi_{k,\theta}\big(\hat{f}_\theta \mid s\big) \Big]. \tag{55}$$

Then the gradient

$$\begin{aligned}\sum_k w_{k,\theta}(s)\nabla_\theta \mathcal{H}(\pi_{k,\theta}) &= -\sum_k w_{k,\theta}(s)\nabla_\theta \mathbb{E}_\epsilon \big[\log \pi_{k,\theta}(\hat{f}_\theta \mid s) \big] \\ &= -\mathbb{E}_{\substack{k\sim \text{Cat}(w_{k,\theta}(s)) \\ \epsilon\sim p(\epsilon)}} \Big[\nabla_\theta \hat{f}_\theta\, \nabla_a \log \pi_{k,\theta}(a \mid s)\big|_{a=\hat{f}_\theta} \Big].\end{aligned} \tag{56}$$

In addition we have

$$\begin{aligned}\sum_k \nabla_\theta w_{k,\theta}(s)\, \mathcal{H}(\pi_k(\cdot|s)) &= -\sum_k w_{k,\theta}(s)\nabla_\theta \log w_{k,\theta}(s) \mathbb{E}_\epsilon \big[\log \pi_{k,\theta}(\hat{f}_\theta \mid s) \big] \\ &= -\mathbb{E}_{\substack{k\sim \text{Cat}(w_{k,\theta}(s)) \\ \epsilon\sim p(\epsilon)}} \big[\nabla_\theta \log w_{k,\theta}(s) \log \pi_k(\hat{f}_\theta \mid s) \big]\end{aligned} \tag{57}$$

$$\begin{aligned}&\int_{s,a} d_{\pi_{m,\theta}}(s)\pi_{m,\theta}(a|s)\nabla_\theta \left(\sum_k w_{k,\theta}(s)\mathcal{H}(\pi_{k,\theta_\theta}(\cdot|s)) \right) da\, ds\,. \\ &= -\mathbb{E}_{\substack{s\sim d_{\pi_{m,\theta}} \\ kk\sim \text{Cat}(w_{k,\theta}(s)) \\ \epsilon\sim p(\epsilon)}} \Big[\nabla_\theta \log w_{k,\theta}(s)\ \log \pi_k\big(\hat{f}_\theta \mid s\big) + \nabla_\theta \hat{f}_\theta\, \nabla_a \log \pi_k(a \mid s)\big|_{a=\hat{f}_\theta} \Big].\end{aligned}$$

Combining these 3 parts with multiplying the last one with $-\alpha$ completes our proof.

□

6.4 Implementation Details of the Policy Iteration Algorithm

To evaluate our policy iteration algorithm, we employed a simple grid-world environment comprising three cell types: walls (impassable), terminal states, and feasible (navigable) states. The agent's objective is to maximize the entropy of its state–action distribution; consequently, it must avoid regions with constrained movement. By definition, actions that would lead into wall cells are excluded from the agent's action set. Likewise, states adjacent to terminal cells are implicitly disfavored, since transitions into a terminal state immediately terminate an episode and prevent further entropy accumulation. As shown in Fig. 4, the highest-probability actions drive the agent toward the center of the upper-right room, where it remains maximally distant from both walls and terminal states, thus preserving the greatest potential for entropy maximization.

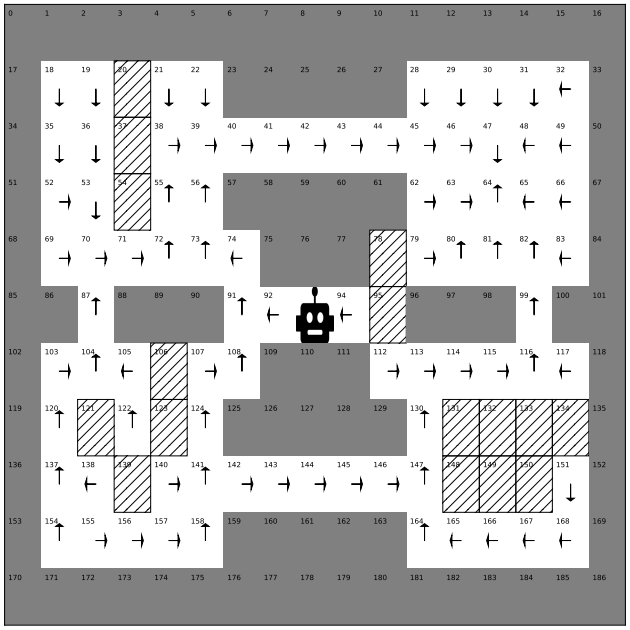

Figure 4: Grid-world environment. Grey cells denote impassable walls, and hatched cells indicate terminal states. Each cell is labeled by its state index. The robot icon marks the agent's initial state. Although the optimal mixture policy is stochastic, arrows show the action with the highest selection probability at each state.

6.5 Implementation Details

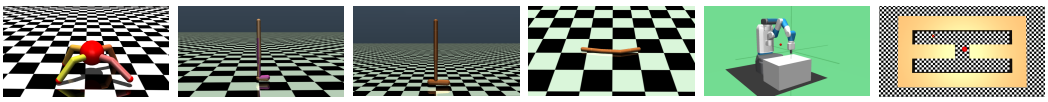

Figure 5: Benchmark environments used to evaluate our framework. From left to right: Ant-v5, Walker2d-v5, Hopper-v5, Swimmer-v5, FetchReach-v2, and AntMaze. These environments span diverse locomotion and manipulation tasks with varying dynamics and control complexity.

Unsupervised Learning: In this section, we detail the implementation of our algorithm for learning mixture components in the continuous action domain. The mixture policy is parameterized by a single neural network with $2K + 1$ output heads. Specifically, K of these heads produce the component means $\mu_i \in \mathbb{R}^M$, another K produce the log–standard deviations $\log \sigma_i \in \mathbb{R}^M$, and the final head produces unnormalized mixing logits $\hat{w} \in \mathbb{R}^K$, from which mixing weights $w \in [0, 1]^K$ are sampled using the Gumbel–Softmax trick, ensuring $\sum_{i=1}^{K} w_i = 1$. The log–standard deviations are exponentiated to obtain positive standard deviations for the Gaussian components, which are then used to form $\mathrm{Normal}(\mu_i, \sigma_i^2)$ distributions.

All component parameters are computed concurrently through a shared feature extractor, with each head responsible for its respective output. Training was conducted to solve the problem defined in Eq. 2. We ran all experiments for 100,000 environment steps, using a discount factor $\gamma = 0.9$ to bias learning toward short-term diversity rather than long-horizon planning. Empirical evaluation demonstrates that this discount factor yields faster convergence to diverse behaviors within the intended temporal scale. To prevent degenerate policies in Ant-v5—where the agent could fall, flip onto its back, and generate high-entropy leg movements in mid-air—we terminate each episode whenever the torso contacts the ground. This termination rule was applied in both the unsupervised variant and the reward-accumulating maze variant, as well as during the four-directional evaluation for our agent and all baselines. We apply the episode termination conditions in the Hopper and Walker environments exactly as specified in their respective Hopper-v5 and Walker-v5 definitions. In

contrast, the Swimmer and Fetch Robot environments were evaluated without any episode termination conditions. For further details see [52].

Downstream Tasks: All methods are trained for 3×10^6 environment steps and evaluated over five random seeds. To ensure fair comparison, health bonuses and control costs are omitted from the reward function. We assess performance on five locomotion tasks: maximizing torso velocity along the $+x$, $-x$, $+y$, and $-y$ axes (directional locomotion), and reaching a central goal in a maze from a random perimeter start (maze navigation) (see Fig. 5). Continuous baselines are trained only on the $+x$ task (using symmetry for other directions), while our method is trained separately on each direction and averaged.

In addition, we train discrete policies using DQN [17] and discrete PPO [16]; to compare with continuous baselines, we also include SAC [15] and PPO [16]. All agents were implemented and trained using the Stable Baselines framework [53], with each algorithm configured according to its recommended hyperparameters.

Compute Resources and Experimental Setup: We document the computational resources and environment used for all experiments, including both unsupervised pretraining and downstream evaluation. Each batch of five agents was trained on a cluster node featuring an NVIDIA T4 GPU, 16 CPU cores, and 4 GB of RAM per core. The CPUs included models such as Intel Xeon and Broadwell. Training time per batch ranged from 30 to 50 hours, while baseline models required less time. All experiments were executed within a Docker container based on the `nvidia/cuda:12.0.0-base-ubuntu20.04` image. Simulations were run using MuJoCo v3.1.6, with full configuration specified in the provided Dockerfile. GPU memory usage during training typically ranged from 8 GB to 10 GB. This setup ensures consistent dependency management and supports full reproducibility of the reported results.

Table 3: Hyperparameters

Parameter	Value
Optimizer	Adam [54]
Learning rate	3×10^{-4}
Discount (γ)	0.9
Replay buffer size	10^5
Number of hidden layers (all networks)	2
Number of hidden units per layer	256
Number of samples per minibatch	32
Steps per epoch	2000
Initial random steps	2000
Nonlinearity	ReLU
Target network smoothing coefficient (τ)	0.005 [55]

6.6 Additional Experimental Results

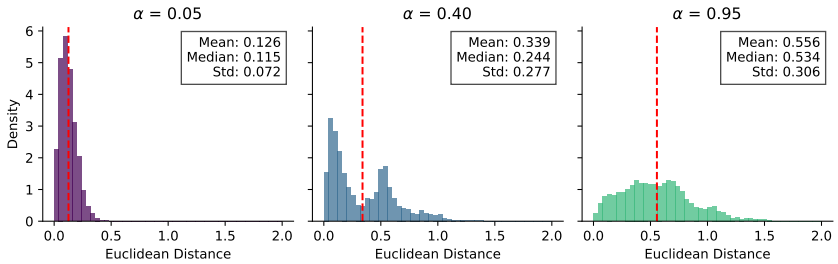

Figure 6: Pairwise Euclidean distance densities between component action means for $\alpha = 0.05, 0.4,$ and 0.95.

Subsequently, we tested the maxi-mix-mini-com entropy objective on the MuJoCo Ant benchmark [23]. For three values of the regularization coefficient ($\alpha = 0.05, 0.4, 0.95$), we trained a 16-component Gaussian mixture policy and measured all pairwise Euclidean distances between the

component mean action vectors under deterministic execution. Figure 6 shows the resulting pairwise distance densities. As α increases, the average distance between components grows, the variation in those distances becomes larger, and the median distance shifts upward, demonstrating that components move farther apart on average and with greater spread. At the same time, each density curve retains density near zero, indicating that many component pairs remain closely clustered even at higher α. Together, these trends show that raising α both expands the policy's overall coverage of the action space and preserves small groups of highly similar, specialized components.

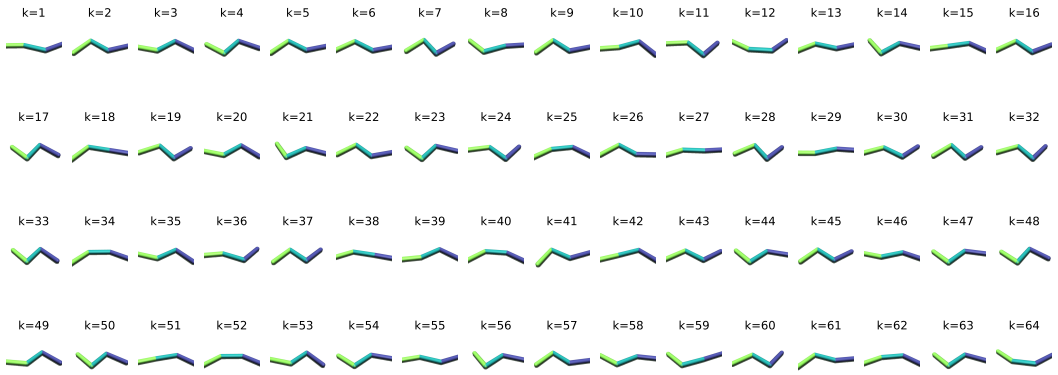

Figure 7: MuJoCo Swimmer-v5 poses resulting from continuously applying the mode of each learned component of a 64-component Gaussian mixture policy as the action. These results illustrate the diversity and coverage of the continuous action space.

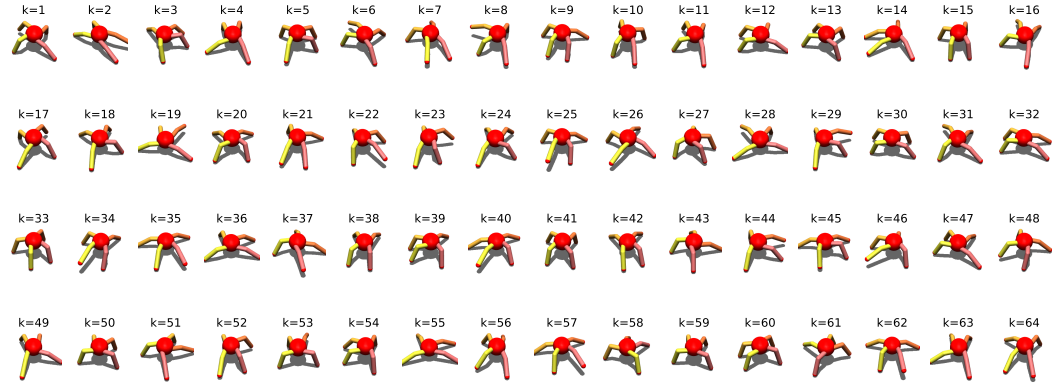

Figure 8: MuJoCo Ant-v5 poses resulting from continuously applying the mode of each learned component of a 64-component Gaussian mixture policy as the action, showing diversity and coverage of the continuous action space.

Algorithm 1 Discrete Policy Updates

Require: A finite state space $\mathcal{S}$, finite action space $\mathcal{A}(s)$, transition kernel $p(s'|s,a)$, discount factor $\gamma \in [0,1)$, specialization parameter $0 \le \alpha < 1$, number of components $K(s)$, number of iterations N. (Sums over a and k are over ranges $0, \cdots, |\mathcal{A}|(s)$ and $1, \cdots, K(s)$).

Ensure: Sequence $\{V^{(n)}, \pi^{(n)}, w^{(n)}, r^{(n)}\}_{n=0}^{N}$ converging to a fixed point.

Initialize:

  $V^{(0)}(s) \leftarrow 0, \quad \forall s \in \mathcal{S}$

  Choose arbitrary $\pi_k^{(0)}(a|s) > 0,\ w_k^{(0)}(s) > 0,\ \forall k, s, a$

  Compute initial mixture policy $\pi_m^{(0)}(a|s) \ = \ \sum_k w_k^{(0)}(s)\, \pi_k^{(0)}(a|s)$

  Compute initial responsibilities $r_k^{(0)}(s,a) \ \leftarrow \ \frac{w_k^{(0)}(s)\, \pi_k^{(0)}(a|s)}{\pi_m^{(0)}(a|s)}, \quad \forall k, s, a$

for $n = 1$ to N **do**

  for all $s \in \mathcal{S}$ **do**

    Compute $Q^{(n-1)}(s,a) \ \leftarrow \ \sum_{s'} p(s'|s,a)\, V^{(n-1)}(s'),\ \forall a$

    $G(s) \ \leftarrow \ G\big(s;\, \pi^{(n-1)}, w^{(n-1)}, r^{(n-1)}\big)$

    $V^{(n)}(s) \ \leftarrow \ G(s) \ + \ \gamma \sum_a \pi_m^{(n-1)}(a|s)\, Q^{(n-1)}(s,a)$

  end for

  for all $k = 1, \ldots, K,\ s \in \mathcal{S},\ a \in \mathcal{A}$ **do**

$$\pi_k^{(n)}(a|s) \ \leftarrow \ \frac{\big[r_k^{(n-1)}(s,a)\big]^{1/(1-\alpha)} \exp\big(\frac{\gamma}{1-\alpha}\, Q^{(n-1)}(s,a)\big)}{\sum_{a'} \big[r_k^{(n-1)}(s,a')\big]^{1/(1-\alpha)} \exp\big(\frac{\gamma}{1-\alpha}\, Q^{(n-1)}(s,a')\big)}$$

  end for

  for all $s \in \mathcal{S},\ k = 1, \ldots, K$ **do**

$$w_k^{(n)}(s) \ \leftarrow \ \frac{\big(\sum_a [r_k^{(n-1)}(s,a)]^{1/(1-\alpha)} \exp \frac{\gamma}{1-\alpha} Q^{(n-1)}(s,a)\big)^{1-\alpha}}{\sum_{l=1}^{K} \big(\sum_a [r_l^{(n-1)}(s,a)]^{1/(1-\alpha)} \exp \frac{\gamma}{1-\alpha} Q^{(n-1)}(s,a)\big)^{1-\alpha}}$$

  end for

  for all $s \in \mathcal{S},\ a \in \mathcal{A},\ k = 1, \ldots, K$ **do**

    Compute new mixture policy $\pi_m^{(n)}(a|s) \ = \ \sum_k w_k^{(n)}(s)\, \pi_k^{(n)}(a|s)$

    Compute new responsibilities $r_k^{(n)}(s,a) \ \leftarrow \ \frac{w_k^{(n)}(s)\, \pi_k^{(n)}(a|s)}{\pi_m^{(n)}(a|s)}$

  end for

end for

Algorithm 2 Learning Components using maxi-mix-mini-com framework for continuous domains

Input: Policy parameters θ, Q-function parameters ϕ_1, ϕ_2, target Q-function parameters $\phi_1^{\text{targ}}, \phi_2^{\text{targ}}$, specialization parameter α, discount factor γ, target update rate ρ, learning rate η, Gumbel–Softmax temperature τ
Initialize replay buffer $\mathcal{D}$
for each iteration **do**
  for each environment step **do**
    Sample component index: $k_t \sim \mathrm{GumbelSoftmax}\big(w_\theta(s_t), \tau\big)$
    Sample action: $a_t \sim \pi_{k_t,\theta}(\cdot|s_t)$
    Execute a_t in the environment; observe next state s_{t+1}, and terminal flag d_t
    Store transition (s_t, a_t, s_{t+1}, d_t) in $\mathcal{D}$
  end for
  for each gradient step **do**
    Sample mini-batch of transitions $\{(s_i, a_i, s_i', d_i)\}_{i=1}^{B} \sim \mathcal{D}$
    Update Q-functions:
    for each sample i in the mini-batch **do**
      Sample component for next state: $k_t' \sim \mathrm{GumbelSoftmax}\big(w_\theta(s_i'), \tau\big)$
      Sample next action: $a_i' \sim \pi^b_{k_i',\theta}(\cdot \mid s_i')$
      Compute target:

$$y_i = \gamma(1 - d_i)\Big[Q_\phi^{\text{targ}}(s_i', a_i') - \log \pi_{m,\theta}(a_i'|s_i') + \alpha \log \pi^b_{k_i',\theta}(a_i'|s_i')\Big]$$

    end for
    Update Q-functions by minimizing:

$$\phi_j \leftarrow \arg\min_{\phi_j} \frac{1}{B} \sum_{i=1}^{B} \Big(Q_{\phi_j}(s_i, a_i) - y_i\Big)^2, \quad j \in \{1, 2\}$$

    Update Policy:
    For each state s_i, sample a component and action using the current policy:

$$k_i \sim \mathrm{GumbelSoftmax}\big(w_\theta(s_i), \tau\big), \quad \tilde{a}_i \sim \pi_{k_i,\theta}(\cdot \mid s_i)$$

    Compute the policy gradient:

$$\nabla_\theta J(\pi_\theta^m) \approx \frac{1}{B} \sum_{i=1}^{B} \nabla_\theta \Big[Q_\phi(s_i, \tilde{a}_i) - \log \pi_{m,\theta}(\tilde{a}_i \mid s_i) + \alpha \log \pi_{k_i,\theta}(\tilde{a}_i|s_i)\Big]$$

    Update policy parameters: $\theta \leftarrow \theta + \eta\, \nabla_\theta J(\pi_\theta^m)$
    Update Target Networks: $\phi_j^{\text{targ}} \leftarrow \rho\, \phi_j^{\text{targ}} + (1 - \rho)\, \phi_j, \quad j \in \{1, 2\}$
  end for
end for