BEHAVIORAL TOKENIZATION VIA MAXIMUM ENTROPY MIXTURE POLICIES WITH MINIMUM ENTROPY COMPONENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Standard continuous-control learning in embodied agents entangles movement with reward, producing task-specific policies but no explicit action vocabulary that another controller can directly reuse. We consider the problem of *behavioral tokenization*: how to learn a fixed, compact bank of state-dependent motor behaviors that becomes a shared discrete interface for future tasks. Previous work has studied this problem using action quantization, sub-policies, options and skills, but no solution provides an unsupervised, task-agnostic method to discover short-lived, action-spanning behavioral tokens. We introduce an online unsupervised framework that discovers behavioral tokens via a joint entropy objective—maximizing the cumulative entropy of an overall mixture policy to ensure rich, exploratory behavior under the maximum occupancy principle, while minimizing the entropy of each component policy to enforce diversity and high specialization. We prove convergence of the value function under our policy-iteration algorithm in the tabular setting, and then extend it to continuous control by fixing the discovered components and deploying them deterministically as quantized actions within an online optimizer to maximize reward. Experiments demonstrate that our *maxi-mix-mini-com* entropy-based tokenization and action quantization provide reusable behavioral tokens that can be sequenced to achieve competitive performance against continuous-action controllers, largely outperform random action quantization and skill discovery methods, and can be fine-tuned to solve downstream tasks showing superior performance to state-of-art off-policy controllers.

1 INTRODUCTION

Standard continuous-control learning in embodied agents entangles movement with a particular reward. The resulting policy is task-specific and provides no explicit action vocabulary that another controller can directly reuse. We study the problem of *behavioral tokenization*: how to learn reward-independent movements through interactions with the environment serving as a fixed, compact bank of state-dependent motor choices that can be reused to solve future tasks. Such a bank is important to separate motor learning from task-specific decision making: downstream RL controllers, planners, or sequence models can then reuse the same frozen, or fine-tuned, components and choose among a small number of behavioral tokens rather than act directly in the continuous space.

Previous work has used action quantization (AQ), sub-policies, options, and skills as ways to build token-like behaviors. However, these fail in one or more fundamental aspects. In AQ methods (Luo et al., 2023; Seo et al., 2025), the aim is to partition the continuous action space into a concise discrete, state-dependent set of actions. Aside from naive per-dimension action binning, which suffers from the curse of dimensionality (Chebotar et al., 2023), current successful methods rely on curated offline datasets (Luo et al., 2023) or expert demonstrations (Dadashi et al., 2022) to build the action set. This approach ties the discovered quantized actions to the dataset's task, hindering both generalization and reusability of the behavioral tokens (Luo et al., 2023). Another direction to behavioral tokenization, which is closest to ours, is to express the policy as a mixture of component sub-policies (Daniel et al., 2016; Seyde et al., 2022). This leads to short-lived (one step) behavioral tokens, but current approaches are task-dependent, making tokens not transferable (Daniel et al., 2016). Further, in

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

existing algorithms, the learned policy collapses to a single component as learning progresses (e.g., Sheebaelhamd et al. (2025)).

Alternative approaches for behavioral tokenization are options and skills methods (Eysenbach et al., 2019; Bacon et al., 2017), but they show limitations regarding effective behavioral tokenization. First, options and skills produce long-lived tokens that must execute for many consecutive timesteps before a new token can be selected, which limits the agent's ability to rapidly interleave different behaviors in response to changing state conditions. Short-lived tokens, by contrast, can be freely recombined at every timestep, enabling fine-grained behavioral composition. Here, we draw an analogy to large language models, which operate on sequences of discrete sub-word tokens (Vaswani et al., 2017)—rather than sentence-level primitives—because fine-grained units enable compositional generalization that sentence-level atoms cannot. Second, options and skills provide a *state-centric* view of behavioral tokenization: behavioral tokens are primarily associated to *where* an agent goes (state marginals, terminal states, or trajectories), not *what* it does. For instance, existing skill discovery methods (Eysenbach et al., 2019; Park et al., 2022; 2024; Sharma et al., 2020) couple a latent variable to states or state trajectories by maximizing their mutual information. In contrast, in an *action-centric* view, diverse behavioral tokens are coupled to the *same* state (e.g, being able to walk and run in different directions from the same location), giving more flexibility to behavioral composability.

Here, we address the above challenges: how to discover in an unsupervised manner state-dependent behavioral tokens that are short-lived and action-centric, and how to compose them to solve any downstream task. Using Markov Decision Processes (MDPs) in discrete time, we define behavioral tokens as one-step component policies of a mixture policy—formally, options with universal initiation and one-step termination. We frame action behavioral tokenization as the problem of learning a finite set of component policies each with minimal action entropy, whose combination into a mixture maximizes cumulative action entropy (Fig. 1a). This *maxi-mix-mini-com entropy* formulation for behavioral tokenization is entirely unsupervised and does not rely on any offline dataset, task-specific definition or demonstrations. The goal of maximizing cumulative action entropy—the maximum occupancy principle (MOP) (Moreno-Bote & Ramirez-Ruiz, 2023)—promotes producing all behaviors compatible with the agent's dynamics and environment, while the goal of minimizing components' entropy enforces diversity and distinctness among tokens (Fig. 1b). The result is a bank of distinct, state-dependent tokens that effectively spans action space and are composable to solve any downstream task (Fig. 1c). Because universally present action bottlenecks due to body non-linearities or terminal states (e.g., agent's damage) reduce action entropy, learned tokens are reusable across tasks and environments and produce span-seeking and safe actions (Fig. 1a).

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). Our algorithm simultaneously learns component policies and mixture weights, and both are generally state-dependent. Building on the tabular case, we extend our theory to the continuous action case (Sec. 3) and provide an unsupervised online AQ method, where quantized actions are the modes of the discovered components. We empirically show that learned quantized actions scale effectively, transferring successfully to novel environment layouts and achieving competitive performance relative to continuous-action PPO (Schulman et al., 2017) in high-dimensional, on-policy control tasks (Sec. 4) with as few as ~ 16 components. Both skill discovery methods and naive random AQ systematically are inferior to our approach. We show that fine-tuning the learned tokens to solve downstream tasks outperforms state-of-art off-policy methods and results in learning and transfer speedups. All in all, our results establish a principled framework for behavioral tokenization and AQ resulting in interpretable components that can be flexibly composed for efficient exploration and generalize well to unseen tasks.

2 MAXI-MIX-MINI-COM ENTROPY OBJECTIVE FOR TOKENIZATION

We aim to learn task-agnostic, short-lived, state-dependent behavioral tokens that facilitate generation of diverse behaviors and the solution of downstream tasks. We first start with the discrete action-state case, and later address the continuous action-state case (Sec. 3). We consider a 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

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

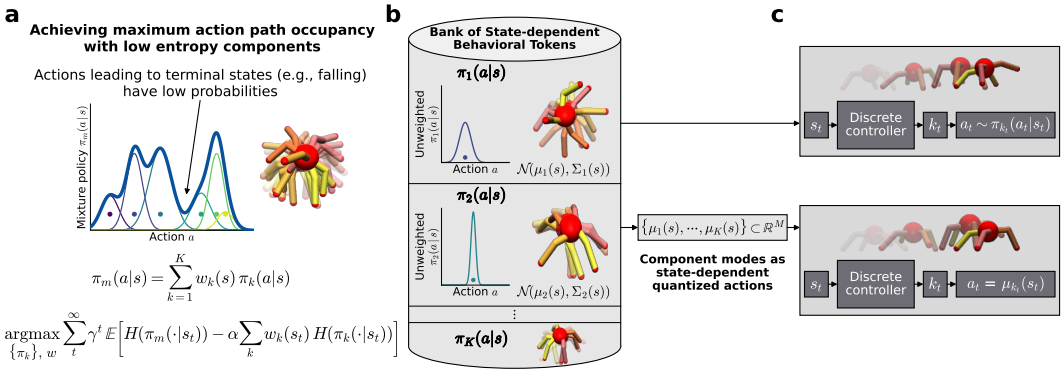

Figure 1: Behavioral token discovery via unsupervised *maxi-mix-mini-com* entropy learning. **(a)** The components of a mixture policy are learned to maximize cumulative action entropy under MOP while minimizing individual entropy (bottom), which entails avoiding falling close to action bottlenecks or terminal states (top). **(b)** The result is a bank of K distinct behavioral tokens (components π_k). In continuous action spaces, we take components to be Gaussian, so learning delivers means $\mu_k(s)$ and diagonal covariance matrices $\Sigma_k(s)$. **(c)** For each downstream task, the components are frozen, the pretraining mixing controller weights are discarded, and a new discrete controller (e.g., DQN) selects among them. The discrete action space is the index set $k \in \{1, \ldots, K\}$, where selecting index k executes component π_k at state s either by sampling an action from $\pi_k(\cdot|s)$ or by using its mode $\mu_k(s)$ to accumulate task-specific reward.

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, and $0 \leq \alpha < 1$. For $\alpha > 0$, 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 produced diversity is also shaped by the state-dependence of the action set $\mathcal{A}(s)$. Indeed, most realistic environments have action bottlenecks (e.g., corridors) and terminal states (e.g., empty battery) where the size of available actions is reduced to a few or just one action (e.g., not moving when dead or having no energy). State-dependence actions sets strongly shape the solution, leading to maximum entropy policies that are non-uniform, promoting exploration and diverse behavior while trying to avoid bottlenecks and terminal states (Ramírez-Ruiz et al., 2024).

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 A.2.1). Importantly, for $0 < \alpha < 1$, our objective promotes distinct, specialized components, and the exact tabular updates prevent component collapse at every finite iteration when initialized with positive weights and non-identical components (see Appendix A.2).

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

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

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

where $Q(s,a)$ is the expected continuation action-value function from the successor state – not to be confused with the conventional Q-action-value function, which also includes immediate reward. Therefore, the objective can be expressed as maximizing the value function over components and mixture weights through 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 $\pi = \{\pi_1, ..., \pi_K\}$, $w = \{w_1, ..., w_K\}$, and 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 greater than or equal to the size of the action space) solving Eq. 5 is equivalent to solving an unconstrained problem with no restrictions on the mixture policy. Although straightforward, this result shows that there is a limit where the problem converges to a known solution.

Theorem 1. *For $0 < \alpha < 1$, if $K(s) \geq |\mathcal{A}(s)|$ for all s, then there is an optimal solution of Eq. 5 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*

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

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. A.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. 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 equations that to our knowledge cannot be solved explicitly (Sec. A.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. Next, we define the immediate gain $G(s;\pi,w,r)$ as

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

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 (Arimoto, 1972). The above definitions and observations allow us to write the optimality Bellman equation 5 as

$$V^*(s) = \max_{\pi,w} \max_r \left(G(s;\pi,w,r) + \gamma \sum_a \pi_m(a|s) Q^*(s,a) \right) \,. \tag{10}$$

As in the Blahut–Arimoto algorithm (Blahut, 1972; Arimoto, 1972) and in more recent applications (Daniel et al., 2016; Arenz et al., 2020), we use this fact to build an algorithm showing monotonic

improvement of the value function until convergence. Our algorithm, described 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

$$r_k^{(0)}(s,a) = w_k^{(0)}(s)\, \pi_k^{(0)}(a|s) / \pi_m^{(0)}(a|s) > 0 \,. \tag{11}$$

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

$$V^{(n)}(s) = G(s; \pi^{(n-1)}, w^{(n-1)}, r^{(n-1)}) + \gamma \sum_a \pi_m^{(n-1)}(a|s) Q^{(n-1)}(s,a) \,, \tag{12}$$

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

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

$$\pi_k^{(n)}(a|s) = \frac{[r_k^{(n-1)}(s,a)]^{\frac{1}{1-\alpha}} e^{\frac{\gamma}{1-\alpha} Q^{(n-1)}(s,a)}}{\sum_{a'} [r_k^{(n-1)}(s,a')]^{\frac{1}{1-\alpha}} e^{\frac{\gamma}{1-\alpha} Q^{(n-1)}(s,a')}} \,. \tag{13}$$

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

$$w_k^{(n)}(s) = \frac{\left(\sum_a [r_k^{(n-1)}(s,a)]^{\frac{1}{1-\alpha}} e^{\frac{\gamma}{1-\alpha} Q^{(n-1)}(s,a)} \right)^{1-\alpha}}{\sum_{k'} \left(\sum_a [r_{k'}^{(n-1)}(s,a)]^{\frac{1}{1-\alpha}} e^{\frac{\gamma}{1-\alpha} Q^{(n-1)}(s,a)} \right)^{1-\alpha}} \,. \tag{14}$$

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

$$r_k^{(n)}(s,a) = w_k^{(n)}(s)\, \pi_k^{(n)}(a|s) / \pi_m^{(n)}(a|s) \,, \tag{15}$$

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. 12,13,14,15 in this order, improves the value function monotonically until convergence.*

Proof. See Appendix Sec. A.2.

2.1 Experiments and results

We first validate Algorithm 1 in a deterministic grid-world with impassable walls and terminal states (see Fig. 4 in Appendix). The action set is {`up`, `right`, `down`, `left`} when away from the walls. Any action that would move into a wall is not available. For $0 < \alpha < 1$, when capacity is large, $K = 4 = \max_s |\mathcal{A}(s)|$, we find that component policies are sharply peaked at a single action, and they span all actions (Fig. 2a), as predicted by Theorem 1. 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 span the full action space. The optimal mixture policy is non-uniform: the weights associated to each component are not equal (Fig. 2a,b). This is because taking an action that steps into a terminal state collapses future entropy to zero, so cells bordering terminals autonomously assign vanishing probability to those transitions despite them remaining admissible, and this effect propagates to parent cells iteratively. The non-uniformity of the optimal mixture policy does not critically depend on having state-dependent primitive action sets, as when all cells have the same number of actions regardless of wall distances the same non-uniformity holds (Fig. 5 in Appendix). In general, policy components with higher weight tend to be broader than the ones that are chosen less frequently (see second component in Fig. 2b). Notably, the agent generates safe component policies in the sense that the probability of falling into a terminal state is close to zero, a fact that strongly shapes the

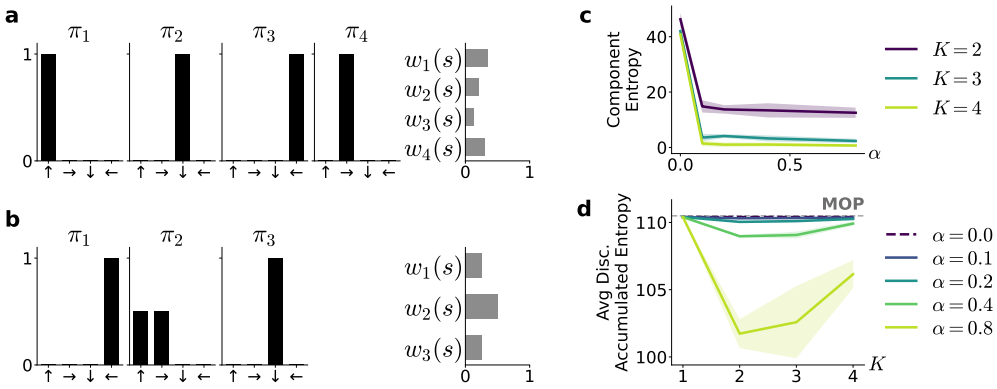

Figure 2: Component specialization and cumulative mixture entropy in a grid world (Fig. 4, Appendix A.5). **(a,b)** Learned state-dependent policies when $K = 4$ (a) and $K = 3$ (b). Policies $\pi_1, \pi_2, \ldots$ (left panels) and mixture weights $w_k(s)$ (right panel) are shown for state $s = 141$ in the grid-world. **(c)** Averaged component entropy decreases as a function of α and for increasing K (different lines), showing enhanced specialization. **(d)** For fixed K, cumulative mixture entropy decreases as α increases; for $\alpha > 0$, adding components beyond $K = 2$ generally recovers mixture entropy toward the MOP limit (different lines). Solid curves: averages over five random seeds; shaded regions: one standard deviation. Dashed line in panel (d) indicates MOP, the theoretical limit.

final mixture policy, making it non-uniform. 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, for fixed K, the cumulative entropy of the mixture policy decreases as α increases, consistent with Theorem 4 in Appendix A.2. For $\alpha > 0$, adding components beyond $K = 2$ generally recovers mixture entropy toward the theoretical limit imposed by MOP (Fig. 2d; dashed line). When $K = 1$, the problem reduces 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. 1c). Next, we analyze the resulting components, quantifying the framework's ability to control action spanning 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 with the Extended Policy Gradient Theorem:

Theorem 3. *Let $\pi_{m,\theta}(a|s)$ be a continuous-action mixture policy with learnable component policies $\pi_{k,\theta}$ and weights $w_{k,\theta}$. Defining $J(\theta) = V(s_0)$ as in Eq. 3, its gradient is*

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

where $d_{\pi_{m,\theta}}$ is the unnormalized discounted occupancy measure defined in Appendix A.3, and $R(s;\pi_\theta, w_\theta)$ is the intrinsic entropy reward in Eq. 5.

Proof. See Appendix Sec. A.3.

For optimization, we extend the half-reparameterization mixture-policy estimator of He et al. (2025) to our reward-free objective, with reparameterized Gaussian components and a straight-through estimator for the discrete mixture weights; full derivation, proof, and pseudocode are in Appendix A.3.

3.1 Experiments and Results

Next, we evaluate behavioral tokenization with the maxi-mix-mini-com objective in continuous action spaces. We first used the MuJoCo Swimmer benchmark (Todorov et al., 2012), which comprises three

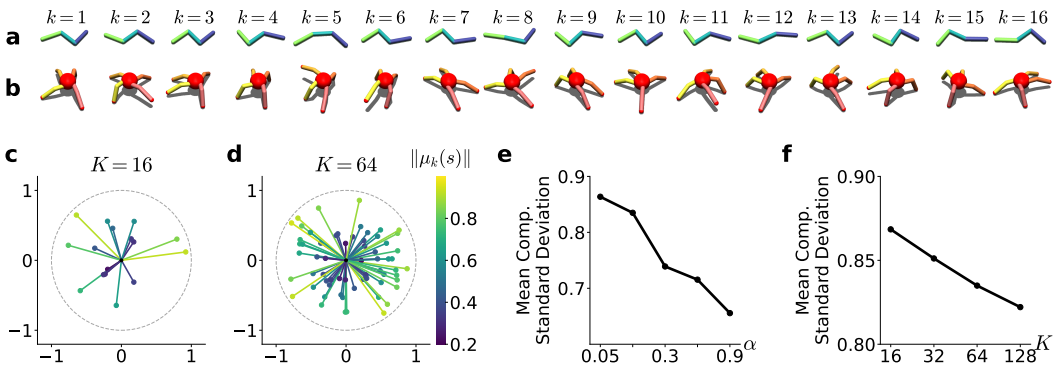

Figure 3: **(a)** Swimmer and **(b)** Ant poses from continuously applying the mode of each learned component as quantized action, showing diversity and action spanning. **(c,d)** Swimmer component action mode vectors (means $\mu_k(s)$ of the Gaussian policies) at a representative random state sampled from a trajectory induced by the mixture policy, for **(c)** $K = 16$ and **(d)** $K = 64$ components. **(e)** Ant ($K = 64$) mean component standard deviation vs. α. **(f)** Ant mean standard deviation vs. K.

segments connected by two rotary joints in a linear chain, and has 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. 11 in Appendix A.7 for $K = 64$; Videos 1). Their diversity and action spanning are better visualized by plotting the modes of each of the Gaussian components, using 16 components (Fig. 3c) or 64 components (Fig. 3d).

Second, we used the MuJoCo Ant benchmark (Todorov et al., 2012), a 3-dimensional quadruped with a central torso and four two-segment legs (8 actuated hinge joints). We consider terminal states as those cases where the torso touches the floor (e.g., a harmful fall). Like the swimmer, learned policies result in interpretable coordinated movements when deployed using their modes (Fig. 3b, for $K = 16$; see Fig. 12 in Appendix A.7 for $K = 64$; Videos 2). The components are diverse and span a broad range of reaching movements with one or two legs, 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 hyperparameters. As expected, the standard deviation averaged over components ($K = 64$) decreases with α (Fig. 3e), indicating an increased degree of component specialization, and increasing K reduces component standard deviation (and thus their entropy) (Fig. 3f). We find broad utilization of the full component bank: for $K = 16$, every component has nonzero mean usage, ranging from 4%–13% (vs. 6.25% under uniform weights). Broad component activation permits invocation and transition among tokens, enabling composition for downstream tasks (Videos 3).

4 Hierarchical Reinforcement Learning with Pre-trained Tokens

Having shown that our objective learns diverse and specialized components, we test whether a frozen bank provides a useful and reusable interface for downstream control (Fig. 1c). To our knowledge, this is the first method to learn online, without task rewards, a reusable bank of state-dependent, one-step actions for quantizing continuous control. We compare our tokenization against complementary alternatives: random action quantization, unsupervised skill discovery, and direct continuous control. For each task, we discard the pretraining controller, freeze all components, and train only a new downstream discrete controller using DQN (Mnih et al., 2013) or PPO (Schulman et al., 2017). At each state, the controller selects a token index k, and we execute the corresponding deterministic component mode $\mu_k(s)$; thus, the K token indices produce K state-dependent continuous actions (see Appendix A.4). This protocol isolates the utility of the learned bank from any task-specific fine-tuning of its components. We evaluate whether the same bank can be reused across tasks, whether learning the bank is necessary, how it compares with conventional skill discovery, how much continuous-control capacity it retains, and how sensitive it is to K and α.

Can one frozen token bank support different downstream tasks? To test whether behavioral tokens form a reusable motor interface rather than a task-specific action set, we pretrain a single $K = 64$ bank

Table 1: Performance ($\pm 95\%$ CI) of our and unsupervised skill discovery algorithms.

Method	Ant (K=64)	Humanoid (K=64)
Ours	$\mathbf{3498.27 \pm 186.08}$	$\mathbf{5672.19 \pm 62.49}$
DADS	124.85 ± 16.64	2516.38 ± 67.30
LSD	1320.54 ± 87.12	1088.69 ± 410.57
DIAYN	341.84 ± 26.02	661.70 ± 97.43
METRA	443.46 ± 121.00	1024.60 ± 49.57

Table 2: Return AUC ($\pm 95\%$ CI; $n = 10$) over 10^5 downstream steps. "Forward" omits the healthy bonus.

Environment	Ours	SAC	Random init.
Ant-v5	$\mathbf{936.7 \pm 91.0}$	597.8 ± 53.3	657.2 ± 63.8
Ant (fwd.)	$\mathbf{332.0 \pm 88.0}$	229.0 ± 86.6	221.1 ± 39.8
Hopper-v5	$\mathbf{867.4 \pm 280.7}$	526.8 ± 130.8	422.4 ± 72.4
Hopper-v5 (fwd.)	$\mathbf{560.0 \pm 192.3}$	276.1 ± 94.6	260.0 ± 40.5
Walker2d-v5	$\mathbf{844.9 \pm 177.4}$	450.3 ± 26.2	481.1 ± 78.0
Walker2d-v5 (fwd.)	$\mathbf{389.0 \pm 105.4}$	147.0 ± 18.5	222.3 ± 47.7

Table 3: Discrete controllers over our frozen tokens largely match continuous-action PPO across tasks (mean return $\pm 95\%$ CI).

Environment	PPO	Ours $K = 16$	Ours $K = 64$
Locomotion (MuJoCo)			
Humanoid-v5	2285.35 ± 948.65	5360.89 ± 108.05	$\mathbf{5672.19 \pm 62.49}$
Ant-v5	3176.44 ± 224.21	2936.96 ± 23.18	$\mathbf{3580.79 \pm 31.87}$
Walker2d-v5	$\mathbf{3834.31 \pm 160.21}$	2394.14 ± 63.61	2473.93 ± 234.04
Hopper-v5	2825.80 ± 176.51	3010.87 ± 145.66	$\mathbf{3226.50 \pm 119.30}$
Swimmer-v5	98.07 ± 6.58	109.96 ± 3.80	$\mathbf{119.05 \pm 3.03}$
Other Continuous Control			
BipedalWalker-v3	281.27 ± 8.57	265.69 ± 15.37	$\mathbf{284.93 \pm 11.96}$
LunarLanderContinuous-v3	217.68 ± 22.45	128.81 ± 35.22	$\mathbf{220.38 \pm 13.06}$
FetchReach-v2	0.80 ± 0.18	$\mathbf{1.00 \pm 0.00}$	$\mathbf{1.00 \pm 0.00}$

on the MuJoCo Ant in an open arena, freeze its components, and reuse the same components across 5 downstream reward objectives: four directional-locomotion tasks, each rewarding velocity along a different cardinal direction, and one goal-directed maze-navigation task from random perimeter starts in a new layout. Directional locomotion requires sustained propulsion along a specified axis, whereas maze navigation requires steering, turning, and composing movements around walls. For each task, only a new downstream controller is trained. We compare against standard continuous PPO (Schulman et al., 2017) (see Appendix A.6 for training details). With $K = 64$, the frozen bank matches continuous PPO on directional locomotion (3276.54 ± 235.23 vs. 3169.83 ± 217.25) and achieves approximately $2.6\times$ its maze-navigation return (9.55 ± 0.71 vs. 3.64 ± 0.57). These results answer the question affirmatively: a single reward-independent bank can serve as a reusable motor interface in downstream tasks without component fine-tuning, including navigation in a layout absent during pretraining.

Is learning the token bank necessary, or is random quantization sufficient? Our objective learns state-dependent components that are individually specialized yet collectively provide diverse actions. We test whether this learned structure is necessary by comparing against static banks of K joint actions sampled uniformly, from a zero-mean Gaussian with standard deviation 0.2, or per dimension from $\{-m, 0, m\}$; at $m = 1$, the nonzero choices coincide with full-range bang–bang controls (Seyde et al., 2021). Our learned bank outperforms the random baselines in most evaluated settings, with particularly large margins on Humanoid and Walker, where random AQ often fails to keep the agent from falling (Tables 7, 8; Videos 4). Thus, static random action sets are not a substitute for a state-dependent bank learned from the embodiment's dynamics.

Can conventional skill-discovery methods serve as one-step behavioral tokens? As discussed above, unsupervised skill-discovery (USD) methods primarily learn temporally extended, state-centric skills: their objectives distinguish states, transitions, or trajectories rather than enforcing a diverse repertoire of actions at the same state. To test whether these skills can nevertheless serve as one-step behavioral tokens, we compare against four leading USD methods—DIAYN, DADS, LSD, and METRA (Eysenbach et al., 2019; Sharma et al., 2020; Park et al., 2022; 2024)—in two representative locomotion environments, Ant-v5 and Humanoid-v5. All banks contain 64 frozen choices, omit the agent's x, y coordinates, and use the same discrete PPO controller trained for 10^7 steps, selecting one deterministic action per environment step. Our method outperforms all four USD baselines in both environments (Table 1), while qualitative comparisons show a broader repertoire of distinct actions (Videos 5; Appendix A.7). Thus, under the matched one-step protocol, action-centric components provide more effective behavioral tokens than the tested state-centric USD methods.

Can a small discrete token bank retain the control capacity of continuous actions? This comparison tests how much control is preserved by tokenization: continuous-action PPO selects

actions directly in an M-dimensional continuous space, whereas our downstream controller chooses among only K frozen, task-agnostic, state-dependent tokens at each step. Across eight continuous-control benchmarks, $K = 64$ attains a mean return at least as high as continuous-action PPO in seven environments, with Walker2d as the only exception (Table 3). Thus, across most of these benchmarks, useful control is retained after replacing continuous action selection with a finite, reusable token interface. On Ant-v5, mean return across α rises 43.1% from $K = 16$ to 64 and 15.2% from 64 to 128; the best α varies with K (Table 8 in Appendix A.7). The benefit is particularly pronounced in difficult-to-stabilize systems: with only $K = 16$ choices, our method achieves more than twice the return of continuous-action PPO on Humanoid, which has 17 action dimensions. We observe a similar advantage in a stabilization-constrained Ant variant, where episodes terminate when the torso leaves a narrow healthy-height band. Under the same downstream training budget, our frozen pretrained tokens achieve a return of 55.81 and a mean episode length of 498.8 steps, whereas PPO achieves a return of 5.32 and a mean episode length of 7.4 steps (Appendix A.7). This level of retained control is particularly notable because our bank is learned without task rewards and frozen before downstream optimization, whereas continuous PPO is trained directly on each task.

Does reward-free pretraining provide a useful initialization for downstream fine-tuning? The preceding experiments freeze the components bank to isolate the learned token interface. Here, we instead fine-tune the complete pretrained mixture policy and compare it with SAC and the same mixture-policy architecture trained from random initialization (see Appendix A.6.3 for the objective and complete training protocol). Because fine-tuning is off-policy, we use SAC (Haarnoja et al., 2018) rather than the on-policy PPO baseline (Schulman et al., 2017) used above. Across Ant, Hopper-v5, and Walker2d-v5, reward-free pretraining yields higher mean return AUC than both baselines (Table 2; Fig. 8 in Appendix A.6.3). The standard returns of Hopper-v5 and Walker2d-v5 include a healthy bonus for every step the agent remains alive. Because reward-free pretraining still observes termination and can therefore learn to avoid falling, we also report a forward score obtained by removing this bonus from the standard return. For Ant, we separately train all methods on a forward-only task. Our method retains its advantage under this metric, showing that the improvement is not explained by the healthy bonus alone. Because the random-initialization baseline uses the same mixture-policy architecture and fine-tuning objective, the improvement reflects the initialization learned during reward-free interaction rather than the architecture alone. Thus, the pretrained bank is useful not only as the source of a frozen discrete token interface but also as an initialization that can adapt to downstream rewards.

5 Discussion and Conclusion

Our paper has tackled an important, yet overlooked, problem: how to generate a bank of behavioral tokens that are task-agnostic, state-dependent, action-centric and learnable in an unsupervised manner. We have presented an online unsupervised method for behavioral tokenization using the maxi-mix-mini-com entropy objective. This objective generates diverse behavioral tokens because the mixture policy aims to maximize cumulative action entropy. The objective also promotes the discovery of distinct tokens because of the per-component entropy penalty. The objective does not generally collapse to uniform action distributions: terminal states and nonlinear dynamics make some actions reduce future entropy, inducing structured, non-uniform mixture policies even without task rewards. We have also provided an unsupervised online AQ algorithm, which takes the modes of the learned components as quantized actions. Our AQ algorithm, paired with discrete-action controllers, performs competitively with continuous-action PPO and outperforms the tested random-quantization and skill-discovery baselines. Separately, fine-tuning the pretrained mixture policy yields higher mean return AUC than SAC on the three tested tasks after 3×10^5 additional reward-free interactions.

Limitations: Our continuous experiments use variance downscaling and deterministic component modes. The method currently fixes K without a principled selection rule. Learning a bank incurs one-time pretraining and must currently be repeated for each embodied agent, although it can be reused across tasks for that same agent. Reuse is not zero-shot: each downstream task requires a new discrete controller. The survival bias in our objective can conflict with downstream goal-reaching tasks that require fast termination, suggesting an important direction for reward-conditioned or fine-tuned token learning.

AI USE STATEMENT

In accordance with the ICLR 2027 AI policy, we used a large language model only to proofread the manuscript, that is, to correct grammar and improve the clarity of text written by the authors. All research ideas, theory, experiments, and analysis are the authors' own. The authors reviewed all edits and take full responsibility for the content of this paper.

REPRODUCIBILITY STATEMENT

Complete proofs are provided in Appendix A.2 and Appendix A.3, and pseudocode is given in Algorithms 1 and 2. The experimental environments, training procedures, hyperparameters, seed counts, and compute resources are documented in Appendix A.5, Appendix A.6, and Table 9. Anonymized source code and the Docker environment are provided in the supplementary material.

REFERENCES

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

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.

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.

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

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, pp. 1726–1734. AAAI Press, 2017.

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

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

Donald Wayne Bushaw. Differential equations with a discontinuous forcing term. Technical Report 469, Experimental Towing Tank, Stevens Institute of Technology, Hoboken, NJ, January 1953.

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.

Yevgen Chebotar, Quan Vuong, Karol Hausman, Fei Xia, Yao Lu, Alex Irpan, Aviral Kumar, Tianhe Yu, Alexander Herzog, Karl Pertsch, Keerthana Gopalakrishnan, Julian Ibarz, Ofir Nachum, Sumedh Anand Sontakke, Grecia Salazar, Huong T. Tran, Jodilyn Peralta, Clayton Tan, Deeksha Manjunath, Jaspiar Singh, Brianna Zitkovich, Tomas Jackson, Kanishka Rao, Chelsea Finn, and Sergey Levine. Q-Transformer: Scalable offline reinforcement learning via autoregressive Q-functions. In *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pp. 3909–3928. PMLR, 2023. URL https://proceedings.mlr.press/v229/chebotar23a.html.

Robert Dadashi, Léonard Hussenot, Damien Vincent, Sertan Girgin, Anton Raichuk, Matthieu Geist, and Olivier Pietquin. Continuous control with action quantization from demonstrations. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 4537–4557. PMLR, 2022. URL https://proceedings.mlr.press/v162/dadashi22a.html.

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

Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. In *International Conference on Learning Representations*, 2019.

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.

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.

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

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.

Jiamin He, Samuel Neumann, Adam White, and Martha White. Investigating mixture policies in entropy-regularized actor-critic. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=LIz0cBmHn5.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with Gumbel-Softmax. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=rkE3y85ee.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015. URL https://arxiv.org/abs/1412.6980.

Diederik P Kingma and Max Welling. Auto-encoding variational Bayes. In *International Conference on Learning Representations*, 2014. URL https://arxiv.org/abs/1312.6114.

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, pp. 128–135. IEEE, 2005.

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

Michael Laskin, Hao Liu, Xue Bin Peng, Denis Yarats, Aravind Rajeswaran, and Pieter Abbeel. Unsupervised reinforcement learning with contrastive intrinsic control. *Advances in Neural Information Processing Systems*, 35:34478–34491, 2022.

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

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*, 32:7867–7878, 2019.

Jianlan Luo, Perry Dong, Jeffrey Wu, Aviral Kumar, Xinyang Geng, and Sergey Levine. Action-quantized offline reinforcement learning for robotic skill learning. In *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pp. 1348–1361. PMLR, 2023. URL https://proceedings.mlr.press/v229/luo23a.html.

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

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.

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.

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)*, pp. 8651–8657. IEEE, 2022.

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

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

Boris T Polyak and Anatoli B Juditsky. Acceleration of stochastic approximation by averaging. *SIAM Journal on Control and Optimization*, 30(4):838–855, 1992.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-Baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

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.

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

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *International Conference on Machine Learning*, pp. 1312–1320. PMLR, 2015.

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

Younggyo Seo, Jafar Uruç, and Stephen James. Continuous control with coarse-to-fine reinforcement learning. In *Proceedings of The 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pp. 2866–2894. PMLR, 2025. URL https://proceedings.mlr.press/v270/seo25a.html.

Tim Seyde, Igor Gilitschenski, Wilko Schwarting, Bartolomeo Stellato, Martin Riedmiller, Markus Wulfmeier, and Daniela Rus. Is bang-bang control all you need? Solving continuous control with Bernoulli policies. *Advances in Neural Information Processing Systems*, 34:27209–27221, 2021.

Tim Seyde, Wilko Schwarting, Igor Gilitschenski, Markus Wulfmeier, and Daniela Rus. Strength through diversity: Robust behavior learning via mixture policies. In *Conference on Robot Learning*, pp. 1144–1155. PMLR, 2022.

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

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

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT Press, 1998.

Yunhao Tang and Shipra Agrawal. Discretizing continuous action space for on-policy optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 5981–5988, 2020.

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, pp. 4131–4138, 2018.

Arash Tavakoli, Mehdi Fatemi, and Petar Kormushev. Learning to represent action values as a hypergraph on the action vertices. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=Xv_s64FiXTv.

Chen Tessler, Guy Tennenholtz, and Shie Mannor. Distributional policy optimization: An alternative approach for continuous control. *Advances in Neural Information Processing Systems*, 32:1350–1360, 2019.

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*, pp. 5026–5033. IEEE, 2012.

Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Hannah Tan, and Omar G. Younis. Gymnasium: A standard interface for reinforcement learning environments. *arXiv preprint arXiv:2407.17032*, 2024. URL https://arxiv.org/abs/2407.17032.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Nino Vieillard, Marcin Andrychowicz, Anton Raichuk, Olivier Pietquin, and Matthieu Geist. Implicitly regularized RL with implicit Q-values. In *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*, volume 151 of *Proceedings of Machine Learning Research*, pp. 1380–1402. PMLR, 2022. URL https://proceedings.mlr.press/v151/vieillard22a.html.

A Appendix

A.1 Related Work

Quantization of Continuous Action Spaces: In control theory, the idea of discretizing (or quantizing) continuous action spaces dates back to Bushaw's work on discontinuous forcing terms (Bushaw, 1953), and was formalized in the "bang–bang" control problem by Bellman et al. (1956). 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 (Tavakoli et al., 2018; Andrychowicz et al., 2020; Vieillard et al., 2022; Tang & Agrawal, 2020). Similarly, Seyde et al. (2021) replace Gaussian policies with factorized Bernoulli "bang–bang" policies, while per-dimension fixed-grid methods discretize each action coordinate and train a task-specific factorized controller (Tang & Agrawal, 2020; Tavakoli et al., 2018; Andrychowicz et al., 2020). These approaches are effective fixed-discretization baselines, but differ from our setting: their policies are trained end-to-end with extrinsic rewards and induce an implicit Cartesian product of per-dimension actions, whereas our unsupervised stage learns a frozen bank of K joint, state-conditioned behavioral tokens. Complementary approaches instead model inter-dimensional dependencies—either via sequential prediction of continuous actions (Metz et al., 2017), distributional policy optimization frameworks (Tessler et al., 2019), learned Q-network structures for high-dimensional control (Sakryukin et al., 2020), or hypergraph representations of action-value functions (Tavakoli et al., 2021).

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 (Luo et al., 2023). Dadashi et al. (2022) proposed AQuaDem, which constructs a state-conditioned discretization of the continuous action space entirely offline using logged human demonstrations before any online reinforcement learning. In offline settings, an action-quantization objective partitions a finite set of actions drawn directly from the dataset; it does not discover or generate novel behaviors. Concretely, for each state s, the algorithm selects a small "codebook" of prototype actions from the empirical action support. As a result, its ability to propose useful or task-optimized actions is strictly bounded by the diversity and quality of the recorded trajectories. This limitation is illustrated in Luo et al. (2023) (their Table 5), where walker2d-medium-replay-v2 — and similarly halfcheetah and hopper — exhibit very poor performance when the offline dataset lacks expert coverage, while performance improves when expert demonstrations are included (walker2d-expert-v2). Therefore, unless high-quality actions are already present in the dataset or the dataset is extremely large, offline quantization alone has limited capacity to produce task-useful action sets.

In contrast, in the online setting considered in our current paper, a quantization method is not confined to a pre-collected action set. The objective shifts from "compressing" a static action distribution to "constructing" a repertoire of actions that are both diverse and high-reward for downstream tasks. Online quantization can actively propose and refine actions through interaction, thereby learning useful actions rather than merely re-quantizing observed behaviors. Q-Transformer (Chebotar et al., 2023) 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 growth 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 (Sheebaelhamd et al., 2025). 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 (Chang et al., 2021). Moreover, there is no guarantee that the mixture-model representation will maintain diverse component utilization (see e.g., Sheebaelhamd et al., 2025). To fill in these gaps, in this work we have introduced the first unsupervised online AQ method, which does not require offline data nor any task definition.

Mixture Policy Representations and Entropy Estimation: Kolchinsky & Tracey (2017) introduce a family of analytic estimators for the differential entropy of mixture distributions, based on pairwise distance functions between components. He et al. (2025) 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. Daniel et al. (2016) 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. Learned components are specific to a task, and therefore they do not generalize to new tasks or environments. Nematollahi et al. (2022) propose a hybrid approach in which a Gaussian mixture dynamical system is first learned from demonstrations to model trajectory distributions of robot skills. Importantly, the half-reparameterization estimator of He et al. (2025) targets the standard entropy-regularized RL objective where policy improvement is driven by extrinsic rewards and a soft value function. Our maxi-mix–mini-com objective, in contrast, is entirely unsupervised: it maximizes mixture entropy while concurrently penalizing component entropies so that the learned tokens remain diverse yet specialized. Theorem 5 adapts the half-reparameterization construction to this reward-free objective, ensuring that mixture-policy gradients remain tractable even when the only learning signal is the opposing pair of intrinsic entropy terms. This distinction guarantees that the resulting components are dynamics- and embodiment-specific primitives agnostic to any downstream reward.

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 (Sutton & Barto, 1998). Goal-conditioned approaches such as like Universal Value Function Approximators (Schaul et al., 2015) train policies to achieve explicit goals but require predefined goal signals, while skill-discovery methods (e.g., DIAYN (Eysenbach et al., 2019), VALOR (Achiam et al., 2018)) 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 (Bacon et al., 2017). 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.

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 (Gottlieb et al., 2013), novelty seeking (Lehman & Stanley, 2011), empowerment (Klyubin et al., 2005; Leibfried et al., 2019), minimizing free energy (Friston et al., 2013), or occupying action-state space (Ramírez-Ruiz et al., 2024). We have focused on the latter because the maximum occupancy principle (MOP) provides a principled way to generate diverse, complex behaviors limited only 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 (Moreno-Bote & Ramirez-Ruiz, 2023). In any event, it is worth exploring in future work different intrinsic motivation objectives for action-policy discretization.

Unsupervised skill discovery (USD) methods commonly learn a latent-conditioned policy $\pi(a|s, z)$ with a skill latent z sampled once per episode, and optimize intrinsic objectives that couple z to states, short transitions, terminal states, or entire trajectories rather than to per-step action choices. For example, DIAYN maximizes $I(S; Z)$ to induce separable state marginals for each skill, VIC maximizes episode-scale dependence $I(S_T; Z|S_0)$ to enlarge the set of reliably reachable terminations, and VALOR decodes skill identity from full trajectories (Eysenbach et al., 2019; Gregor et al., 2017; Achiam et al., 2018). Other methods explicitly target short-horizon dynamics: DADS maximizes $I(S'; Z|S)$ via a skill-conditioned dynamics model, and CIC formulates a contrastive objective on single-step transitions while regularizing transition entropy to preserve coverage (Sharma et al., 2020; Laskin et al., 2022). Recent work has incorporated geometric or metric structure into the dependency measure (e.g., enforcing Lipschitz/non-expansive embeddings or using Wasserstein dependency measures) to bias learned skills toward dynamic, far-reaching behaviors (Park et al., 2022; 2024). Because these objectives encourage distinguishability at the level of states, transitions, terminations, or trajectories, distinct skills can select identical atomic actions in the same state and remain differentiable only by their subsequent evolution; consequently, USD methods are by design temporally-extended primitives sampled per episode and do not enforce per-timestep action-level

diversity, which limits their suitability for tasks that require tightly interleaved, within-state switching between atomic actions.

In summary, a recurring limitation of prevailing USD objectives is that they couple the latent skill primarily to *where* an agent goes (state marginals or terminal states) rather than to *how* it behaves at shared states (characteristic action patterns, magnitudes, or variability). Therefore, USD objectives are *state-centric*, while our maxi-mix-mini-com objective is *action-centric*. Operationally, for example, *running* toward two different goal states may instantiate the same skill with differing directions, whereas *walking* and *running* along a common heading should be treated as different skills because their one-step effects (increment magnitudes and variability) differ systematically despite substantial state overlap. Objectives that maximize dependence between skills and states can therefore separate by geography or heading without ensuring that skills remain distinguishable when trajectories overlap. For reuse and composition, it is therefore desirable to require identifiability from local behavior under shared state distributions rather than solely from global or terminal state statistics. Our maxi-mix-mini-com entropy-based algorithm accomplishes this goal: it leads to the discovery of action-spanning, state-dependent, single-step, distinguishable and composable behavioral tokens.

A.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

$$V^*(s) = \max_{\pi,w} \left(R(s;\pi,w) + \gamma \sum_a \pi_m(a|s) Q^*(s,a) \right) \tag{16}$$

$$R(s;\pi,w) \equiv \mathcal{H}(\pi_m(\cdot|s)) - \alpha \sum_k w_k(s) \mathcal{H}(\pi_k(\cdot|s)) \,, \tag{17}$$

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 the weighted average entropy of its components – 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 Haarnoja et al., 2018, 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. 16 is equivalent to solving an unconstrained problem with no restrictions on the mixture policy.

Theorem 1. *For $0 < \alpha < 1$, if $K(s) \geq |\mathcal{A}(s)|$ for all s, then there is an optimal solution of Eq. 16 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*

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

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

Proof. Under the assumption $0 < \alpha < 1$, 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. 18. 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 Ramírez-Ruiz et al. (2024). □

Remark. The theorem establishes existence rather than characterizing every optimal decomposition. The constructed solution is deterministic and action-spanning, but zero-weight components in other optimal decompositions are unconstrained because they affect neither π_m nor the objective. At $\alpha = 0$, the objective depends only on π_m, so a deterministic action-covering decomposition can still be chosen, but stochastic, duplicate, or collapsed decompositions inducing the same optimal mixture are also optimal.

For the case $K(s) < |\mathcal{A}(s)|$, we can write the optimality Bellman equation 16 (see Sec. 2) as

$$V^*(s) = \max_{\pi,w} \max_{r} \left(G(s;\pi,w,r) + \gamma \sum_a \pi_m(a|s) Q^*(s,a) \right) \tag{19}$$

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

$$r_k^{(0)}(s,a) = \frac{w_k^{(0)}(s)\, \pi_k^{(0)}(a|s)}{\pi_m^{(0)}(a|s)} > 0\,. \tag{20}$$

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

$$V^{(n)}(s) = G(s;\pi^{(n-1)}, w^{(n-1)}, r^{(n-1)}) + \gamma \sum_a \pi_m^{(n-1)}(a|s) Q^{(n-1)}(s,a)\,, \tag{21}$$

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

$$\pi_k^{(n)}(a|s) = \frac{[r_k^{(n-1)}(s,a)]^{\frac{1}{1-\alpha}} e^{\frac{\gamma}{1-\alpha} Q^{(n-1)}(s,a)}}{\sum_{a'} [r_k^{(n-1)}(s,a')]^{\frac{1}{1-\alpha}} e^{\frac{\gamma}{1-\alpha} Q^{(n-1)}(s,a')}}\,. \tag{22}$$

This equation is the result of the optimization

$$\pi^{(n)}(\cdot|s) = \arg\max_{\pi} \left(G(s;\pi, w^{(n-1)}, r^{(n-1)}) + \gamma \sum_{k,a} w_k^{(n-1)}(s)\, \pi_k(a|s) Q^{(n-1)}(s,a) \right)\,, \tag{23}$$

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 A.2.1). Note that the ordering of the component policies $\pi_1, \ldots, \pi_K$ is immaterial, as the objective in Eq. 23 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. For $0 \leq \alpha < 1$, positive $w_k^{(n-1)}(s)$ and $r_k^{(n-1)}(s,a)$, Eq. 22 is the unique global maximizer of the fixed-V, w, r policy update in Eq. 23, as the solution lies in the interior of the simplex and the objective is strictly concave.

Defining $V_{\pi}^{(n)}$ as the argument of the right hand side of Eq. 23, after replacing π with the optimal $\pi_k^{(n)}(a|s)$, we obtain

$$V_{\pi}^{(n)} = G(s;\pi^{(n)}, w^{(n-1)}, r^{(n-1)}) + \gamma \sum_{k,a} w_k^{(n-1)}(s)\, \pi_k^{(n)}(a|s) Q^{(n-1)}(s,a) \tag{24}$$

$$= (1-\alpha) \sum_k w_k^{(n-1)}(s) \log \left[\sum_a [r_k^{(n-1)}(s,a)]^{\frac{1}{1-\alpha}} e^{\frac{\gamma}{1-\alpha} Q^{(n-1)}(s,a)} \right] - \sum_k w_k^{(n-1)}(s) \log(w_k^{(n-1)}(s)) \,. \tag{25}$$

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. 23 cannot decrease.

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

$$w_k^{(n)}(s) = \frac{\left(\sum_a [r_k^{(n-1)}(s,a)]^{\frac{1}{1-\alpha}} e^{\frac{\gamma}{1-\alpha} Q^{(n-1)}(s,a)} \right)^{1-\alpha}}{\sum_l \left(\sum_a [r_l^{(n-1)}(s,a)]^{\frac{1}{1-\alpha}} e^{\frac{\gamma}{1-\alpha} Q^{(n-1)}(s,a)} \right)^{1-\alpha}} \,. \tag{26}$$

This equation is the result of the optimization

$$w^{(n)}(s) = \arg\max_{w} \left(G(s;\pi^{(n)}, w, r^{(n-1)}) + \gamma \sum_{k,a} w_k(s)\, \pi_k^{(n)}(a|s) Q^{(n-1)}(s,a) \right) , \tag{27}$$

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. 26 is the only global maximum provided that $r_k^{(n-1)}(s,a) > 0$ for all k, s and a (see Appendix A.2.3 for details).

Defining $V_{\pi,w}^{(n)}$ as the argument of the right hand side of Eq. 27, after replacing w with the optimal $w_k^{(n)}(s)$, we obtain

$$V_{\pi,w}^{(n)} = G(s;\pi^{(n)}, w^{(n)}, r^{(n-1)}) + \gamma \sum_{k,a} w_k^{(n)}(s)\, \pi_k^{(n)}(a|s) Q^{(n-1)}(s,a) \tag{28}$$

$$= \log \left[\sum_k \left(\sum_a [r_k^{(n-1)}(s,a)]^{\frac{1}{1-\alpha}} e^{\frac{\gamma}{1-\alpha} Q^{(n-1)}(s,a)} \right)^{1-\alpha} \right] . \tag{29}$$

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. 27 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

$$r_k^{(n)}(s,a) = \frac{w_k^{(n)}(s)\, \pi_k^{(n)}(a|s)}{\pi_m^{(n)}(a|s)} \,, \tag{30}$$

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 A.2.3 for details),

$$r^{(n)}(s,\cdot) = \arg\max_{r} \left(G(s;\pi^{(n)}, w^{(n)}, r) + \gamma \sum_{k,a} w_k^{(n)}(s)\, \pi_k^{(n)}(a|s) Q^{(n-1)}(s,a) \right) , \tag{31}$$

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

Defining $V^{(n)}_{\pi,w,r}$ as the quantity within the bracket on the right hand side of Eq. 31 after replacing r with the optimal $r^{(n)}_k(s,a)$, we obtain

$$V^{(n)}_{\pi,w,r} = G(s;\pi^{(n)},w^{(n)},r^{(n)}) + \gamma \sum_{k,a} w^{(n)}_k(s)\,\pi^{(n)}_k(a|s) Q^{(n-1)}(s,a)\;. \tag{32}$$

Naturally, we have $V^{(n)}_{\pi,w,r}(s) \geq V^{(n)}_{\pi,w}(s)$ for all s, because after the optimization the right hand side of Eq. 31 cannot decrease. This, in turn, implies that $V^{(n)}_{\pi,w,r}(s) \geq V^{(n)}(s)$ 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. 21,22,26,30 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. 21 we see that

$$V^{(n+1)}(s) = G(s;\pi^{(n)},w^{(n)},r^{(n)}) + \gamma \sum_a \pi^{(n)}_m(a|s) Q^{(n)}(s,a) \tag{33}$$

$$\geq G(s;\pi^{(n)},w^{(n)},r^{(n)}) + \gamma \sum_a \pi^{(n)}_m(a|s) Q^{(n-1)}(s,a) \tag{34}$$

$$= V^{(n)}_{\pi,w,r} \tag{35}$$

$$\geq V^{(n)}(s)\;, \tag{36}$$

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 $V^{(n)}_{\pi,w,r}$ in Eq. 32, 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 strict 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. 22,26,30, 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, $\pi^{(0)}_k(a|s) > 0$, $w^{(0)}_k(s) > 0$ and $r^{(0)}_k(s,a) > 0$ 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. 22,26,30. 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. 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.

Remark: For exact tabular updates with $0 < \alpha < 1$, our algorithm prevents both trivial-collapse scenarios at every finite iteration when initialized with positive weights and non-identical components. In both trivial-collapse scenarios—the case where all component policies are identical and the case where the mixture weights are deterministic—the mixture distribution π_m coincides with a single component k, so $\mathcal{H}(\pi_m(\cdot|s)) = \mathcal{H}(\pi_k(\cdot|s))$ and $\sum_r w_r\,\mathcal{H}(\pi_r(\cdot|s)) = \mathcal{H}(\pi_k(\cdot|s))$. Substituting into Eq. 17 yields the same degenerate reward

$$R(s;\pi,w) = \mathcal{H}(\pi_k(\cdot|s)) - \alpha\mathcal{H}(\pi_k(\cdot|s)) = (1-\alpha)\,\mathcal{H}(\pi_k(\cdot|s)),$$

which is strictly lower than the reward under a truly diverse mixture (high $\mathcal{H}(\pi_m)$, low $\sum_k w_k\,\mathcal{H}(\pi_k)$), so the objective favors the diverse mixture over these collapsed solutions. Under the stated initialization, all finite iterates retain positive weights and non-identical components, so neither trivial-collapse scenario occurs.

A.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 $\sum_a \pi_k(a|s) = 1$. 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 $\sum_a \pi_k(a|s) = 1$, 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. 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

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

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

$$\frac{\partial \mathcal{L}_k}{\partial \pi_k(a|s)} = \log r_k^{(n-1)}(s,a) + \gamma Q^{(n-1)}(s,a) - (1-\alpha)\Big(\log\pi_k(a|s) + 1\Big) + \lambda_k = 0, \tag{37}$$

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. 13.

Because $r_k^{(n-1)}(s,a) > 0$ and $Q^{(n-1)}(s,a) < \infty$, the critical point is in the simplex. Therefore $\pi_k^{(n)}$ is the unique global maximizer. 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).

A.2.2 FINDING OPTIMAL MIXTURE WEIGHTS AT ITERATION n

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

$$w_k^{(n)}(s) = \arg\max_w \Big[G\big(s;\, \pi^{(n)}, w, r^{(n-1)}\big) + \gamma \sum_{k,a} w_k(s)\, \pi_k^{(n)}(a|s)\, Q^{(n-1)}(s,a) \Big], \quad \text{subject to} \quad \sum_k w_k(s) = 1, \tag{38}$$

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. 25) as

$$\begin{aligned} &G\big(s;\, \pi^{(n)}, w, r^{(n-1)}\big) + \gamma \sum_{k,a} w_k(s)\, \pi_k^{(n)}(a|s)\, Q^{(n-1)}(s,a) \\ &= (1-\alpha) \sum_k w_k(s) \log F_k^{(n-1)}(s) - \sum_k w_k(s) \log w_k(s)\ , \end{aligned} \tag{39}$$

where

$$F_k^{(n-1)}(s) = \sum_a [r_k^{(n-1)}(s,a)]^{\frac{1}{1-\alpha}}\, e^{\frac{\gamma}{1-\alpha}\, Q^{(n-1)}(s,a)}\ . \tag{40}$$

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

$$\mathcal{L}\big(s, w, \lambda_w\big) = (1-\alpha) \sum_{k=1}^K w_k(s)\ \log F_k^{(n-1)}(s) - \sum_{k=1}^K w_k(s)\ \log w_k(s) + \lambda_w \Big(\sum_{k=1}^K w_k(s) - 1 \Big)\,. \tag{41}$$

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

$$-\log w_k(s) - 1 + (1-\alpha)\ \log F_k^{(n-1)}(s) + \lambda_w = 0. \tag{42}$$

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

$$w_k^{(n)}(s) = \frac{\big(F_k^{(n-1)}(s)\big)^{\,1-\alpha}}{\sum_l \big(F_l^{(n-1)}(s)\big)^{\,1-\alpha}} = \frac{\Big(\sum_a [r_k^{(n-1)}(s,a)]^{\frac{1}{1-\alpha}}\, e^{\frac{\gamma}{1-\alpha} Q^{(n-1)}(s,a)}\Big)^{1-\alpha}}{\sum_l \Big(\sum_a [r_l^{(n-1)}(s,a)]^{\frac{1}{1-\alpha}}\, e^{\frac{\gamma}{1-\alpha}\, Q^{(n-1)}(s,a)}\Big)^{1-\alpha}}\,. \tag{43}$$

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.

A.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

$$\mathcal{L}_r = \sum_{k=1}^{K} w_k^{(n)}(s)\, \pi_k^{(n)}(a|s)\, \log r_k(a|s) + \lambda_r(s,a) \left(1 - \sum_{k=1}^{K} r_k(s,a)\right). \tag{44}$$

Taking the derivative of the Lagrangian in (44) 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).

A.2.4 Finite-iteration separation of tabular components

A natural concern is that repeated policy updates could cause initially distinct components to become identical. The following result rules this out at every finite iteration of the exact tabular algorithm by tracking differences in their relative log-action preferences.

Lemma 1 (Finite-Iteration Separation of Components)**.** *Fix a state s, two components k, ℓ, and two actions $a, b \in \mathcal{A}(s)$. Define the signed difference in their pairwise log-action preferences by*

$$D_{k\ell}^{(n)}(a, b; s) = \log \frac{\pi_k^{(n)}(a|s)}{\pi_k^{(n)}(b|s)} - \log \frac{\pi_\ell^{(n)}(a|s)}{\pi_\ell^{(n)}(b|s)}.$$

Under the positive initialization of Theorem 2, for every finite iteration $n \geq 1$ and every $0 \leq \alpha < 1$,

$$D_{k\ell}^{(n)}(a, b; s) = \frac{D_{k\ell}^{(n-1)}(a, b; s)}{1-\alpha} = \frac{D_{k\ell}^{(0)}(a, b; s)}{(1-\alpha)^n},$$

$$\left|D_{k\ell}^{(n)}(a, b; s)\right| = \frac{\left|D_{k\ell}^{(0)}(a, b; s)\right|}{(1-\alpha)^n}.$$

The sign is preserved. The magnitude is unchanged when $\alpha = 0$ and, for every nonzero difference, grows by a factor $1/(1-\alpha) > 1$ when $0 < \alpha < 1$. Thus, initially distinct components at s remain distinct at every finite iteration.

Proof. Positivity, established in Theorem 2, ensures that all logarithms below are well-defined. Taking the ratio of the updates for actions a and b in Eq. 22 gives, for $i \in \{k, \ell\}$,

$$\log \frac{\pi_i^{(n)}(a|s)}{\pi_i^{(n)}(b|s)} = \frac{1}{1-\alpha} \left[\log \frac{r_i^{(n-1)}(s,a)}{r_i^{(n-1)}(s,b)} + \gamma\big(Q^{(n-1)}(s,a) - Q^{(n-1)}(s,b)\big)\right].$$

Subtracting the expressions for k and ℓ eliminates the common continuation-value term:

$$D_{k\ell}^{(n)}(a, b; s) = \frac{1}{1-\alpha} \log \frac{r_k^{(n-1)}(s,a)\, r_\ell^{(n-1)}(s,b)}{r_k^{(n-1)}(s,b)\, r_\ell^{(n-1)}(s,a)}.$$

Using Eq. 30, and Eq. 20 when $n = 1$, the mixture weights and mixture probabilities cancel, yielding

$$\frac{r_k^{(n-1)}(s,a)\, r_\ell^{(n-1)}(s,b)}{r_k^{(n-1)}(s,b)\, r_\ell^{(n-1)}(s,a)} = \frac{\pi_k^{(n-1)}(a|s)\, \pi_\ell^{(n-1)}(b|s)}{\pi_k^{(n-1)}(b|s)\, \pi_\ell^{(n-1)}(a|s)}.$$

Hence,

$$D_{k\ell}^{(n)}(a,b;s) = \frac{D_{k\ell}^{(n-1)}(a,b;s)}{1-\alpha}.$$

Iterating this identity and taking absolute values proves both formulas. Finally, two distinct strictly positive distributions must differ in at least one pairwise probability ratio. Thus, if the components are initially distinct at s, some $D_{k\ell}^{(0)}(a,b;s)$ is nonzero and remains nonzero at every finite iteration. □

Remark: The growing quantity is a log-preference difference, not a distance between action distributions. For $0 < \alpha < 1$, nonzero differences diverge as $n \to \infty$: action probabilities can approach zero while remaining positive at every finite iteration. This does not force every component to become deterministic. Components can even approach the same limiting policy while their vanishing probabilities decay at different rates. Thus, also when $K > |\mathcal{A}(s)|$, the lemma permits duplicate limiting components and does not preclude vanishing mixture weights.

A.2.5 Information-theoretic interpretation and the role of α

The maxi-mix-mini-com objective combines high mixture entropy with low average component entropy. This encourages broad action coverage and low-entropy components, but does not directly show whether different component indices correspond to distinguishable action choices. We therefore rewrite the objective using mutual information, revealing how α balances action coverage with token distinguishability and how this balance changes at global optima.

At a fixed state s, let C denote the selected component, with $\Pr(C = k \mid S = s) = w_k(s)$, and let A denote the resulting action, with $\Pr(A = a \mid C = k, S = s) = \pi_k(a|s)$. The marginal distribution of A is the mixture policy $\pi_m(\cdot|s)$, while $\mathcal{H}(A \mid C, S = s) = \sum_k w_k(s)\mathcal{H}(\pi_k(\cdot|s))$. Therefore,

$$\begin{aligned} R(s;\pi,w) &= \mathcal{H}(A \mid S = s) - \alpha\mathcal{H}(A \mid C, S = s) \\ &= (1-\alpha)\mathcal{H}(A \mid S = s) + \alpha I(A; C \mid S = s), \end{aligned} \tag{45}$$

$$I(A; C \mid S = s) = \sum_k w_k(s) D_{\mathrm{KL}}(\pi_k(\cdot|s) \,\|\, \pi_m(\cdot|s)) \,.$$

Thus, the first term favors a broad action distribution, while the second favors component policies that select distinguishable actions at the same state. In particular, the information term is zero if and only if all components with positive weight at s have identical action distributions.

To see why this distinction matters, consider two equally weighted components and two actions at a fixed state. If both components choose both actions uniformly, the mixture policy is uniform and has high entropy, but the component index provides no information about the action, so $I(A; C|s) = 0$. If instead one component deterministically chooses the first action and the other chooses the second, the mixture policy remains uniform, but now $I(A; C|s) = \log 2$. Thus, the two decompositions generate the same diverse behavior, but for $\alpha > 0$ our objective favors the one whose component indices represent distinct action choices.

We next show how α controls this trade-off for the full discounted objective, including changes in the states visited by the agent. For an initial-state distribution ρ, define

$$\mathcal{H}_m^\gamma(\pi, w) = \mathbb{E}_{\rho,\pi_m}\left[\sum_{t=0}^{\infty} \gamma^t \mathcal{H}(\pi_m(\cdot|s_t))\right],$$

$$\mathcal{H}_c^\gamma(\pi, w) = \mathbb{E}_{\rho,\pi_m}\left[\sum_{t=0}^{\infty} \gamma^t \sum_k w_k(s_t)\mathcal{H}(\pi_k(\cdot|s_t))\right],$$

$$\mathcal{I}^\gamma(\pi, w) = \mathbb{E}_{\rho,\pi_m}\left[\sum_{t=0}^{\infty} \gamma^t I(A_t; C_t \mid S_t = s_t)\right] = \mathcal{H}_m^\gamma(\pi, w) - \mathcal{H}_c^\gamma(\pi, w).$$

The objective averaged over $s_0 \sim \rho$ is $J_\alpha(\pi, w) = \mathcal{H}_m^\gamma(\pi, w) - \alpha\mathcal{H}_c^\gamma(\pi, w)$.

Using $\mathcal{I}^\gamma = \mathcal{H}_m^\gamma - \mathcal{H}_c^\gamma$, the objective can equivalently be written as

$$J_\alpha = (1-\alpha)\mathcal{H}_m^\gamma + \alpha\mathcal{I}^\gamma.$$

Thus, α controls the balance between broad behavior under the mixture policy and components that select distinguishable actions. The following theorem makes this trade-off precise over complete discounted trajectories, including changes in the states visited by the policy. At global optima, increasing α cannot increase cumulative component entropy or mixture entropy, and cannot decrease the information that the selected component provides about the action. Therefore, α acts as a specialization parameter with a predictable effect on the learned token bank.

Theorem 4 (Effect of Specialization Strength at Global Optima)**.** *Let $0 \le \alpha_1 < \alpha_2 < 1$ and consider the same feasible class for both objectives. Assume that each J_{α_i} attains a global maximum on this class, and let $(\pi^{(i)}, w^{(i)})$ be any global maximizer of J_{α_i}, for $i = 1, 2$. Assume that the discounted entropy sums are finite, and write $\mathcal{H}_{m,i}$, $\mathcal{H}_{c,i}$ and $\mathcal{I}_i$ for the corresponding quantities. Then*

$$\mathcal{H}_{c,2} \le \mathcal{H}_{c,1}, \qquad \mathcal{H}_{m,2} \le \mathcal{H}_{m,1}, \qquad \mathcal{I}_2 \ge \mathcal{I}_1.$$

Moreover, either all three quantities are unchanged, or component entropy strictly decreases and component–action information strictly increases.

Proof. Optimality of each solution at its corresponding value of α gives

$$\mathcal{H}_{m,1} - \alpha_1\mathcal{H}_{c,1} \ge \mathcal{H}_{m,2} - \alpha_1\mathcal{H}_{c,2},$$
$$\mathcal{H}_{m,2} - \alpha_2\mathcal{H}_{c,2} \ge \mathcal{H}_{m,1} - \alpha_2\mathcal{H}_{c,1}.$$

Let $D = \mathcal{H}_{c,1} - \mathcal{H}_{c,2}$. Adding these inequalities gives $(\alpha_2 - \alpha_1)D \ge 0$, and hence $D \ge 0$. Rearranging each inequality separately gives $\alpha_1 D \le \mathcal{H}_{m,1} - \mathcal{H}_{m,2} \le \alpha_2 D$. By the definition of $\mathcal{I}_i$,

$$(\mathcal{H}_{m,1} - \mathcal{H}_{m,2}) + (\mathcal{I}_2 - \mathcal{I}_1) = D,$$

and therefore

$$(1-\alpha_2)D \le \mathcal{I}_2 - \mathcal{I}_1 \le (1-\alpha_1)D.$$

If $D = 0$, these bounds force $\mathcal{H}_{m,1} = \mathcal{H}_{m,2}$, and the identity then gives $\mathcal{I}_1 = \mathcal{I}_2$. If $D > 0$, then $\mathcal{H}_{c,2} < \mathcal{H}_{c,1}$ and $\mathcal{I}_2 - \mathcal{I}_1 \ge (1-\alpha_2)D > 0$, completing the proof. □

As α approaches 1, the objective places relatively more weight on the mutual information between the selected component and the action. The theorem shows that this information cannot decrease between global optima, but it does not guarantee that every component remains active during training.

Consistent with this ordering, Fig. 2d shows that, for every evaluated K, cumulative mixture entropy decreases as α increases.

Remark. The proof provides a sharper quantitative characterization. Let $D = \mathcal{H}_{c,1} - \mathcal{H}_{c,2} \ge 0$. Then

$$\underbrace{\mathcal{H}_{c,1} - \mathcal{H}_{c,2}}_{\text{reduction in component entropy}} = \underbrace{\mathcal{H}_{m,1} - \mathcal{H}_{m,2}}_{\text{loss of mixture diversity}} + \underbrace{\mathcal{I}_2 - \mathcal{I}_1}_{\text{gain in token distinguishability}}.$$

Thus, the reduction in component entropy is divided between lost mixture diversity and increased token distinguishability.

A.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 Sutton & Barto (1998) 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*

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

where $d_{\pi_{m,\theta}}(s) = \sum_{t=0}^{\infty}\gamma^t \Pr(s_t = s|s_0, \pi_{m,\theta})$ is the unnormalized discounted occupancy measure, $\nabla_\theta R(s;\pi_\theta, w_\theta) = \nabla_\theta \mathcal{H}\big(\pi_{m,\theta}(\cdot|s)\big) - \alpha\nabla_\theta \sum_{k=1}^{K} w_{k,\theta}(s)\,\mathcal{H}(\pi_{k,\theta}(\cdot|s))$, $\pi_\theta = \{\pi_{1,\theta}, ..., \pi_{K,\theta}\}$ and $w_\theta = \{w_{1,\theta}, ..., w_{K,\theta}\}$

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

$$\begin{aligned}
&\nabla_\theta V(s)\\
&= \nabla_\theta R(s;\pi_\theta, w_\theta) + \gamma\nabla_\theta \int_a \pi_{m,\theta}(a|s)Q(s,a)\, da\\
&= \nabla_\theta R(s;\pi_\theta, w_\theta) + \gamma \int_a \nabla_\theta \pi_{m,\theta}(a|s)Q(s,a)\, da + \gamma \int_a \pi_{m,\theta}(a|s)\nabla_\theta Q(s,a)\, da\\
&= \nabla_\theta R(s;\pi_\theta, w_\theta) + \gamma \int_a \nabla_\theta \pi_{m,\theta}(a|s)Q(s,a)\, da + \gamma \int_a \pi_{m,\theta}(a|s)\nabla_\theta \int_{s'} p(s'|s,a)V(s')\, ds'\, da\\
&= \nabla_\theta R(s;\pi_\theta, w_\theta) + \gamma \int_a \nabla_\theta \pi_{m,\theta}(a|s)Q(s,a)\, da + \gamma \int_{a,s'} \pi_{m,\theta}(a|s)p(s'|s,a)\nabla_\theta V(s')\, ds'\, da\\
&= \nabla_\theta R(s;\pi_\theta, w_\theta) + \gamma \int_a \nabla_\theta \pi_{m,\theta}(a|s)Q(s,a)\, da +\\
&+ \gamma \int_{a,s'} \pi_{m,\theta}(a|s)p(s'|s,a)\\
&\Big(\nabla_\theta R(s';\pi_\theta, w_\theta) + \gamma \int_{a'} \nabla_\theta \pi_{m,\theta}(a'|s')Q(s',a')\, da'\\
&\quad +\gamma \int_{a',s''} \pi_{m,\theta}(a'|s')p(s''|s',a')\nabla_\theta V(s'')\, ds''\, da'\Big)\\
&= \int_x \sum_{t=0}^{\infty}\gamma^t \Pr(s \to x, t, \pi_{m,\theta}) \left(\nabla_\theta R(x;\pi_\theta, w_\theta) + \gamma \int_a \nabla_\theta \pi_{m,\theta}(a|x)Q(x,a)\right) dx
\end{aligned} \tag{46}$$

where we define $Pr(s \to x, t, \pi_{m,\theta})$ similar to Sutton & Barto (1998) as the probability of transitioning from state s to state x in t steps under the mixture policy $\pi_{m,\theta}$. Then we can find that

$$\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta V(s_0)\\
&= \int_s \sum_{t=0}^{\infty}\gamma^t \Pr(s_0 \to s, t, \pi_{m,\theta}) \left(\nabla_\theta R(s;\pi_\theta, w_\theta) + \gamma \int_a \nabla_\theta \pi_{m,\theta}(a|s)Q(s,a)\right) ds\\
&= \int_{s,a} d_{\pi_{m,\theta}}(s)\pi_{m,\theta}(a|s)\left(\nabla_\theta \log \pi_{m,\theta}(a|s)\gamma Q(s,a) + \nabla_\theta R(s;\pi_\theta, w_\theta)\right)
\end{aligned} \tag{47}$$

□

In this work, we reparametrize component actions as $a = f_\theta(\varepsilon; s, k)$, where $\varepsilon \sim p$ and the resulting action follows $\pi_{k,\theta}(\cdot|s)$. The following theorem gives the corresponding on-policy gradient identity; the replay-based update in Algorithm 2 is an approximation.

Theorem 5 (Component-Reparameterization Policy Gradient Theorem)**.** *Let $\hat{f}_\theta = f_\theta(\varepsilon; s, k)$ then the gradients of objective function under our reparametrization become*

$$\begin{aligned}
&\nabla_\theta J\big(\pi_{m,\theta}\big) \\
&= \int_s d_{\pi_{m,\theta}}(s)\, \mathbb{E}_{\substack{k\sim Cat(w_{k,\theta}(s)) \\ \varepsilon\sim p}} \\
&\quad \Big[\nabla_\theta \log w_{k,\theta}(s)\, \Big(\gamma Q_{\pi_{m,\theta}}\big(s, \hat{f}_\theta\big) - \log \pi_{m,\theta}\big(\hat{f}_\theta|s\big) + \alpha \log \pi_{k,\theta}\big(\hat{f}_\theta|s\big)\Big) \\
&\quad +\ \nabla_\theta \hat{f}_\theta\ \nabla_a \Big(\gamma Q_{\pi_{m,\theta}}(s,a) - \log \pi_{m,\theta}(a|s) + \alpha \log \pi_{k,\theta}(a|s)\Big)\Big|_{a=\hat{f}_\theta}\Big]\, ds,
\end{aligned} \tag{48}$$

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

Proof. Our proof depends heavily on a similar proof provided by He et al. (2025), 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

$$\nabla_\theta J(\theta) = \int_{s,a} d_{\pi_{m,\theta}}(s)\pi_{m,\theta}(a|s)\left(\nabla_\theta \log \pi_{m,\theta}(a|s)\gamma Q(s,a) + \nabla_\theta R(s;\pi_\theta, w_\theta)\right)\, da\, ds\,. \tag{49}$$

We can break this last equation into 3 parts that we can handle separately. The first part is for the continuation-value term, which recovers the vanilla policy gradient theorem (Sutton & Barto, 1998). He et al. (2025) found that

$$\begin{aligned}
&\int_{s,a} d_{\pi_{m,\theta}}(s)\pi_{m,\theta}(a|s)\nabla_\theta \log \pi_{m,\theta}(a|s)\gamma Q_{\pi_{m,\theta}}(s,a)\, da\, ds \\
&= \int_s d_{\pi_{m,\theta}}(s)\, \mathbb{E}_{\substack{k\sim \text{Cat}(w_{k,\theta}(s)) \\ \epsilon\sim p(\epsilon)}} \\
&\quad \Big[\gamma Q_{\pi_{m,\theta}}\big(s,\ \hat{f}_\theta\big)\ \nabla_\theta \log w_{k,\theta}(s) \\
&\quad + \gamma \nabla_\theta \hat{f}_\theta\ \nabla_a Q_{\pi_{m,\theta}}(s,a)\big|_{a=\hat{f}_\theta}\Big]\, ds.
\end{aligned} \tag{50}$$

The second part handles the entropy of the mixture policy and similarly He et al. (2025) found that:

$$\int_{s,a} d_{\pi_{m,\theta}}(s)\, \pi_{m,\theta}(a|s)\, \nabla_\theta \mathcal{H}\big(\pi_{m,\theta}(\cdot|s)\big)\, da\, ds \tag{51}$$

$$= -\int_s d_{\pi_{m,\theta}}(s)\, \mathbb{E}_{\substack{k\sim \text{Cat}(w_{k,\theta}(s)) \\ \epsilon\sim p(\epsilon)}} \Big[\begin{matrix}\log \pi_{m,\theta}\big(\hat{f}_\theta|s\big)\ \nabla_\theta \log w_{k,\theta}(s) \\ +\ \nabla_\theta \hat{f}_\theta\ \nabla_a \log \pi_{m,\theta}(a|s)\big|_{a=\hat{f}_\theta}\end{matrix}\Big]\, ds. \tag{52}$$

The third term is

$$\int_{s,a} d_{\pi_{m,\theta}}(s)\pi_{m,\theta}(a|s)\nabla_\theta \left(\sum_k w_{k,\theta}(s)\mathcal{H}(\pi_{k,\theta}(\cdot|s))\right)\, da\, ds\,. \tag{53}$$

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

$$\mathcal{H}(\pi_{k,\theta}(\cdot|s)) = -\,\mathbb{E}_{\epsilon\sim p(\epsilon)}\Big[\log \pi_{k,\theta}\big(\hat{f}_\theta|s\big)\Big]. \tag{54}$$

Then the gradient

$$\begin{aligned}
\sum_k w_{k,\theta}(s)\nabla_\theta \mathcal{H}(\pi_{k,\theta}) &= -\sum_k w_{k,\theta}(s)\nabla_\theta \mathbb{E}_\epsilon\big[\log \pi_{k,\theta}(\hat{f}_\theta|s)\big] \\
&= -\mathbb{E}_{\substack{k\sim \text{Cat}(w_{k,\theta}(s)) \\ \epsilon\sim p(\epsilon)}}\Big[\nabla_\theta \hat{f}_\theta\ \nabla_a \log \pi_{k,\theta}(a|s)\big|_{a=\hat{f}_\theta}\Big].
\end{aligned} \tag{55}$$

In addition we have

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

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

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

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

□

Differentiable mixture-weight sampling. The categorical mixture weights cannot be directly reparameterized; we therefore sample them with the straight-through Gumbel-Softmax estimator (Jang et al., 2017), which yields a biased but differentiable sample. Given Gumbel noise g_k and temperature τ, the differentiable soft sample is

$$y_k = \frac{\exp((\log w_{k,\theta}(s) + g_k)/\tau)}{\sum_j \exp((\log w_{j,\theta}(s) + g_j)/\tau)}.$$

The forward pass uses $\hat{z} = \text{one_hot}(\arg\max_k(\log w_{k,\theta}(s) + g_k))$, the backward pass uses $z = \hat{z} + y - \text{StopGrad}(y)$, and the differentiable action is $a = \sum_k z_k f_\theta(\varepsilon; s, k)$, where $f_\theta(\varepsilon; s, k)$ is the reparameterized component sample (Kingma & Welling, 2014). Pseudocode is given in Algorithm 2.

A.4 Mode-Based vs. Stochastic Component Execution

The behavioral tokens in our method are the component policies $\pi_k(\cdot|s)_{k=1}^K$ themselves; the discrete high-level controller selects a token index $k \in 1, \ldots, K$ at each timestep. To execute this selection as a continuous control signal, there are two natural grounding strategies: either sample an action $a_t \sim \pi_k(\cdot|s_t)$ from the selected component, or execute the deterministic mode $a_t = \mu_k(s_t)$ of that component. In all experiments reported in the main text we adopt the mode-based approach for three practical reasons. First, using modes yields a one-to-one mapping $(s, k) \mapsto a$ and thereby induces a standard discrete-action MDP, which allows us to directly employ off-the-shelf discrete RL algorithms (DQN, discrete PPO) and enables fair comparison with random quantization and unsupervised skill discovery methods which are also applied deterministically. Second, using modes removes the additional stochasticity that would arise from sampling $\pi_k(\cdot|s)$, reducing variance in value estimation and isolating the effect of the learned token set rather than conflating it with per-step sampling noise. Third, our maxi-mix-mini-com objective explicitly drives each component $\pi_k(\cdot|s)$ toward low entropy and strong specialization via the $-\alpha \sum_k w_k(s_t)\mathcal{H}(\pi_k(\cdot|s_t))$ penalty (Eq. 2); in this low-entropy regime the mode $\mu_k(s)$ is a faithful summary of the component's behavior.

Importantly, nothing in our framework prevents us from using the stochastic variant—i.e., having the discrete policy select k and then sampling $a_t \sim \pi_k(\cdot|s_t)$ for execution. This simply corresponds to a hierarchical policy over one-step stochastic skills rather than over deterministic quantized actions, and may be preferable in domains where preserving exploration noise at the low level is beneficial. We leave systematic comparison of these two grounding strategies to future work.

A.5 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.

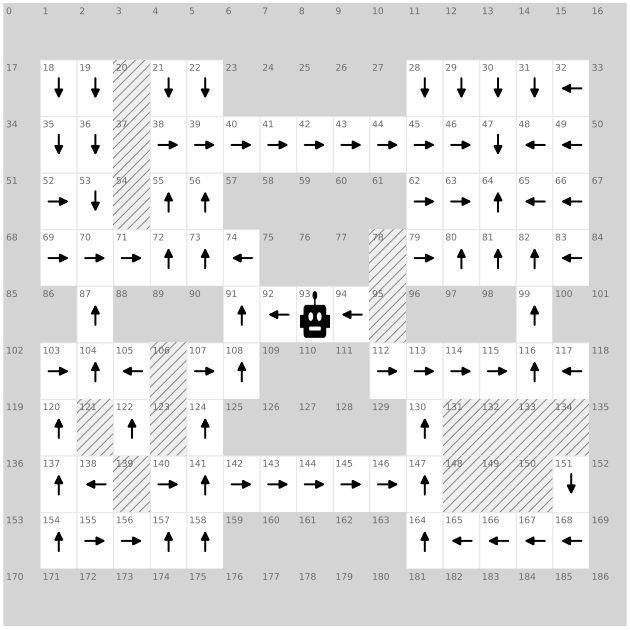

Figure 4: Grid-world environment. Grey cells denote impassable walls, and hatched cells indicate terminal states. Each cell is labeled by its state index. The robot icon marks the agent's initial state. Although the optimal mixture policy is stochastic, arrows show the action with the highest selection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

An immediate concern is whether the same behavior would collapse to a uniform distribution if every state, including those adjacent to walls or terminal cells, were forced to retain the full action set. Figures 5.a and 5.b reproduce the analysis of state 141 from Fig. 4 with four- and three-component mixtures, respectively, while now enforcing all actions even near walls; despite this imposed symmetry, both mixtures still shift probability mass away from moves that approach terminal states. Figure 5.c shows the solution for the grid world in Fig. 4 under the same constraint. The resulting mixture remains structured: actions that lead back toward terminal regions are systematically downweighted rather than averaged out. Consequently, even when admissible action sets are equalized across states, the entropy-seeking policy does not collapse to a uniform distribution, but instead continues to favor trajectories that prolong future entropy accumulation.

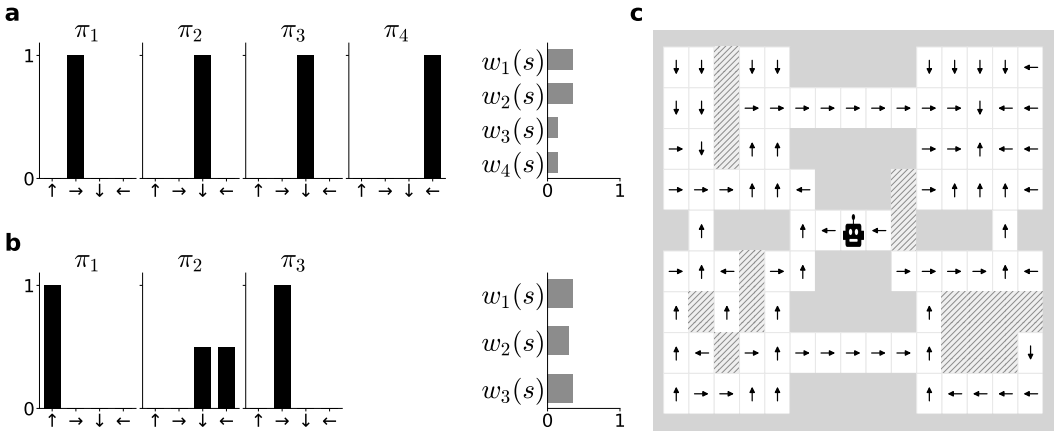

Figure 5: Effect of enforcing identical action sets for wall-adjacent states. (a,b) repeats the analysis of state 141 with four- and three-component mixtures, respectively, while retaining all actions even near walls; the resulting distributions remain non-uniform and favor moves away from terminal cells. (c) presents the mixture policy—constructed as in the example of Fig. 4 but with all actions retained even near walls—and shows that the resulting distribution maintains a structured preference that steers the agent away from terminal regions.

A.6 Implementation Details for Continuous Control

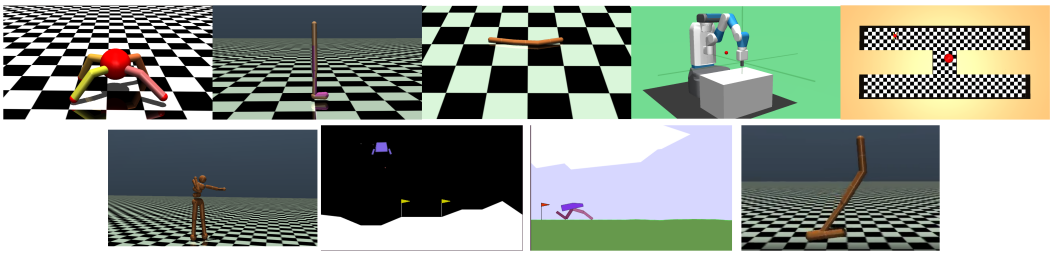

Figure 6: Benchmark environments used to evaluate our framework. From left to right: Ant-v5, Walker2d-v5, Swimmer-v5, FetchReach-v2, AntMaze, Humanoid-v5, BipedalWalker-v3, LunarLanderContinuous-v3 and Hopper-v5. These environments span diverse locomotion and manipulation tasks with varying dynamics and control complexity.

A.6.1 Unsupervised component learning

In this section, we detail the implementation of our algorithm for learning mixture components in the continuous action domain. The mixture policy uses a shared network with two linear output layers that produce the component means $\mu_i \in \mathbb{R}^M$ and log–standard deviations $\log \sigma_i \in \mathbb{R}^M$ for all K components. A linear pretraining controller produces mixing logits $\hat{w} \in \mathbb{R}^K$, which are converted to normalized weights $w \in [0,1]^K$ by softmax. During training, a component is sampled from these weights using the straight-through Gumbel–Softmax estimator. Each log standard deviation is clipped to $[-20, 2]$ before exponentiation. For component k, we sample $\epsilon \sim \mathcal{N}(0, I)$, set $z = \mu_k(s) + \sigma_k(s)\epsilon$, and execute the normalized action $a = \tanh(z) \in (-1, 1)^M$, without further rescaling or hard clipping. The corresponding component log density includes the tanh change-of-variables correction,

$$\log \pi_k(a|s) = \log \mathcal{N}\big(z; \mu_k(s), \sigma_k^2(s)\big) - \sum_{j=1}^{M} \log\big(1 - a_j^2\big).$$

For numerical stability, we evaluate each correction term from the pre-tanh value as $\log(1 - \tanh^2 z_j) = 2\big(\log 2 - z_j - \mathrm{softplus}(-2z_j)\big)$. The mixture entropy is evaluated using the true

marginal density $\log \pi_m(a|s) = \mathrm{logsumexp}_k\big(\log w_k(s) + \log \pi_k(a|s)\big)$. Writing C for the component index, for $0 \le \alpha \le 1$ the intrinsic reward satisfies $R(s) = (1-\alpha)\mathcal{H}(A|s) + \alpha I(A;C|s) \le (1-\alpha)M \log 2 + \alpha \log K$; hence $V(s)$ is bounded above by this quantity divided by $1-\gamma$. For deterministic evaluation and diagnostics, the pretraining controller selects $k^* = \arg\max_k w_k(s)$ and executes $a = \tanh(\mu_{k^*}(s))$ without sampling noise; for downstream tasks, the discrete controller instead selects k and executes $a = \tanh(\mu_k(s))$.

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 experiments for Swimmer-v5 and FetchReach-v2 for 100,000 environment steps, for Ant-v5 for 200,000 environment steps, and for Humanoid-v5 and Walker2d-v5 for 2,000,000 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 Humanoid-v5 and Walker2d-v5 environments exactly as specified in their respective definitions. In contrast, the Swimmer and FetchReach (robot) environments were evaluated without any episode termination conditions. See our hyperparameters in Table 9. For further details, see Towers et al. (2024).

A.6.2 Downstream control with frozen tokens

For the Ant reuse experiment reported in Section 4, all downstream controllers are trained for 1×10^7 environment steps and evaluated over 10 random seeds. To ensure a fair comparison, healthy bonuses and control costs are omitted from the reward function. We evaluate five downstream 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. 6). The continuous PPO baseline is trained only on the $+x$ task (using symmetry for other directions), while our method is trained separately on each direction and averaged. For these five Ant tasks, we pretrain the bank of behavioral tokens once in the standard, open Ant arena (without maze walls) using only the agent's proprioceptive observation vector s_t comprising joint angles, joint velocities, and body-centric features. Downstream directional controllers therefore act in the same open layout, whereas the maze-navigation controller encounters a new layout with walls; in both cases the pretrained components remain frozen. The discrete controller always receives the proprioceptive state s_t, and only the maze task augments that input with achieved and desired goals $(s_t, g_t^{\text{ach}}, g^{\text{des}})$ following the standard goal-conditioned RL formulation. In contrast, the low-level components never observe goal variables or layout-specific signals and continue to act solely on s_t. For the results in Table 1, we use the official implementation of Park et al. (2024) to train the unsupervised skill discovery (USD) method after omitting the x and y coordinates from the state space. All USD methods were trained for 3×10^6 steps. We then trained discrete PPO (Schulman et al., 2017) on top of our components and on the USD methods for 1×10^7 steps. The same configuration is used for the results in Table 7, except that for FetchReach-v2 we use DQN (Mnih et al., 2013) due to its empirically superior performance on that environment. Thus, Ant, Humanoid, Swimmer, Walker2D, Hopper, Bipedal, LunarLander and AntMaze all employ discrete PPO as the high-level controller, whereas FetchReach-v2 uses DQN.

Hyperparameter Selection Protocol. The reward-free component-learning experiments use the optimizer, learning rate, batch size, and network architecture listed in Table 9. We vary the specialization weight α and the number of components K over a small grid, and report downstream performance for each setting in Table 8. The separate fine-tuning settings are given in Table 4.

For downstream RL, we use the Stable-Baselines3 implementations (Raffin et al., 2021): continuous PPO serves as the continuous-action baseline, while discrete PPO and DQN act as high-level controllers over behavioral token banks. For a given task, we keep the discrete controller's network, optimizer settings, and training budget fixed when comparing our components with random action alphabets or unsupervised skill baselines. The unsupervised skill-discovery baselines use the official METRA release of Park et al. (2024) with its recommended hyperparameters, without additional sweeps.

A.6.3 FINE-TUNING THE PRETRAINED MIXTURE POLICY

The frozen-token experiments above test whether the learned components form a useful discrete interface. We additionally test whether reward-free pretraining provides a useful initialization when the complete mixture policy is allowed to adapt to a downstream reward.

Objective. Fine-tuning combines the task action-value with the same maxi-mix-mini-com terms used during pretraining. The mixture policy maximizes

$$\mathcal{J}_{\beta,\alpha}(\theta) = \mathbb{E}_{s\sim\mathcal{D}}\left[\sum_{k=1}^{K} w_{k,\theta}(s)\,\mathbb{E}_{\epsilon\sim\mathcal{N}(0,I)}\Big[Q_\phi(s,a_k) + \beta\big(-\log\pi_{m,\theta}(a_k|s) + \alpha\log\pi_{k,\theta}(a_k|s)\big)\Big]\right], \tag{57}$$

where $a_k = \tanh(\mu_{k,\theta}(s) + \sigma_{k,\theta}(s)\epsilon)$, $\mathcal{D}$ is the replay buffer, and β controls the strength of the maxi-mix-mini-com terms relative to the task value. The twin critics are trained using the target

$$y = r + \gamma(1-d)\left[\min_{j\in\{1,2\}} Q^{\text{targ}}_{\phi_j}(s',a') + \beta\left(-\log\pi_{m,\theta}(a'|s') + \alpha\log\pi_{k',\theta}(a'|s')\right)\right], \tag{58}$$

where $k' \sim \text{Cat}(w_\theta(s'))$ and $a' \sim \pi_{k',\theta}(\cdot|s')$. The terminal flag d denotes true termination; time-limit truncations bootstrap. For these experiments, we evaluate the sum over all K components in Eq. 57 exactly, using one reparameterized action sample per component, rather than sampling a single component with a straight-through estimator.

Architecture. The mixture-policy network has a three-layer, 256-unit shared encoder and a two-layer, 32-unit component network. Linear output layers produce the component parameters, while the pretraining controller maps the shared representation to K mixing logits. Each of the two critics is a three-layer, 256-unit network receiving the state and action. The mixture-policy network uses LayerNorm, ReLU, and dropout 0.1; the critics use ReLU.

Training phases. Pretraining learns the mixture policy with zero environment rewards but retains the environment's termination conditions. Fine-tuning loads the complete pretrained mixture policy, including the shared encoder, component network, all component heads, and pretraining controller. It initializes a new twin critic, restores the task reward, and jointly updates all mixture-policy parameters. Thus, we neither fine-tune one token at a time nor replace the mixture with SAC's single Gaussian policy. Instead, we continue off-policy training of the full mixture policy. Table 4 summarizes the settings for both phases.

Table 4: Training protocol for the fine-tuning experiment. Both phases use $K = 64$, Adam, batch size 32, discount $\gamma = 0.99$, and Polyak coefficient 0.995.

Setting	Reward-free pretraining	Downstream fine-tuning
Environment steps	3×10^5	10^5
Reward	0	Task reward
Mixture-policy initialization	Random	Pretrained mixture policy
Critic initialization	Random	Random
Intrinsic-objective weight β	1	0.2 (0 for Ant-v5 full return)
Specialization α	0.1 (Ant, Walker2d-v5); 0.5 (Hopper-v5)	0.1
Pretraining controller learning rate	10^{-3}	10^{-3}
Encoder/component learning rate	10^{-3}	10^{-4}
Critic learning rate	10^{-3}	10^{-3}
Initial exploration	2,000 uniform steps	2,000 steps, 20% uniform

Baselines. The random-initialization baseline uses the same $K = 64$ mixture-policy architecture, objective, and downstream schedule of 10^5 environment steps, but receives no pretraining. SAC uses a single tanh-squashed Gaussian policy, the same twin-critic architecture, learning rate 10^{-3}, and $10{,}000$ initial uniform-action steps.

Evaluation. We evaluate each method deterministically on 20 fixed episodes every $5{,}000$ steps. We define return AUC as the mean return over checkpoints from $5{,}000$ through $100{,}000$ steps, excluding the zero-shot checkpoint. For Hopper-v5 and Walker2d-v5, each method is trained on standard return and evaluated on both standard and forward scores, with the latter obtained by subtracting the $+1$ healthy bonus per step; our runs use $\beta = 0.2$. For Ant-v5, all methods have separate standard-return and forward-only runs. Our standard-return run uses $\beta = 0$, which yielded higher AUC than $\beta = 0.2$, while our forward-only run uses $\beta = 0.2$. We analyze below how removing the intrinsic term affects the fine-tuned controller's token choices (Table 5). Each result averages 10 independent runs, with 95% confidence intervals computed using the Student t distribution. For our method, one run comprises an independently pretrained mixture policy and one fine-tuning run. The methods therefore receive the same 10^5 task-reward environment steps, while our method additionally receives 3×10^5 reward-free environment steps.

Hyperparameter selection. We fix $K = 64$ throughout. Fine-tuning uses $\alpha = 0.1$ and $\beta = 0.2$, except for Ant-v5 standard return ($\beta = 0$). For SAC, we select the best forward-reward AUC from temperatures $\{0.05, 0.1, 0.2, 0.5\}$, yielding 0.05, 0.2, and 0.5 for Ant, Hopper-v5, and Walker2d-v5, respectively.

A.6.4 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 10 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. The main setup uses MuJoCo v3.1.6; FetchReach-v2 uses a separate MuJoCo v2.3.7 setup specified in `requirements-fetchreach.txt`. GPU memory usage during training typically ranged from 8 GB to 10 GB.

A.6.5 Licenses for existing assets:

Our experiments use standard simulated-control environments and open-source RL implementations. MuJoCo is licensed under Apache License 2.0, while Gymnasium, Gymnasium-Robotics, Stable-Baselines3, and the official METRA implementation are licensed under the MIT license. We cite the corresponding papers or software packages in the main text and appendix, and the supplementary code preserves the relevant third-party license notices and dependency information.

A.7 Additional Experimental Results

Learning dynamics of discrete vs. continuous controllers. Figure 7 plots episodic return as a function of environment interactions for continuous-action PPO and discrete controllers operating over our learned behavioral tokens with $K = 16$ and $K = 64$ components on the same continuous-control benchmarks used in Table 3. The discrete controller is PPO except on FetchReach-v2, where we use DQN. For each method, we plot the mean over 10 random seeds and show 95% confidence intervals, where individual seed curves (except for FetchReach-v2) are first exponentially smoothed with coefficient $\alpha = 0.9$. The discrete controllers, which operate on fixed banks of unsupervised components, exhibit stable learning behavior and converge to a significant fraction of continuous PPO performance on all tasks, with larger K generally improving final returns. These results support the claim that a modest number of learned behavioral components is sufficient to support strong downstream control in a variety of continuous-control benchmarks.

Fine-tuning dynamics and preservation of token diversity. Figure 8 shows the downstream learning curves corresponding to Table 2. To determine whether fine-tuning preserves the learned token bank, we additionally measure token usage and action diversity at fine-tuning step 0, before any downstream update, and at step $100{,}000$, averaged over the same 10 random seeds used for the return results. For pretrained rows, step 0 is the loaded pretrained policy; for random-initialization rows, it is the untouched random policy. Statistics are computed on states visited by the current policy at each evaluation, so the state distribution may change between the two measurements.

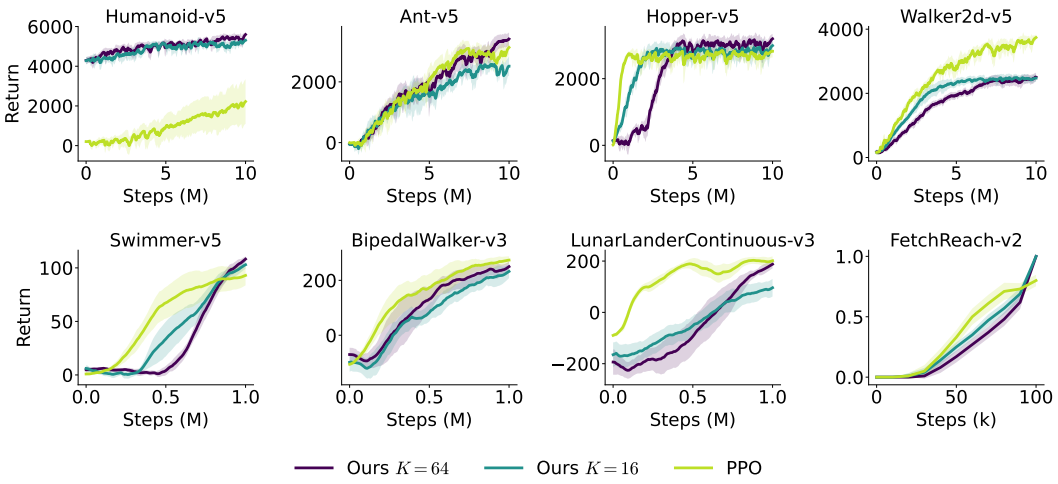

Figure 7: Learning curves for our method with $K \in \{16, 64\}$ frozen behavioral components compared to the continuous-action PPO baseline on the continuous-control benchmarks reported in Table 3. For each method, curves show the mean episodic return over 10 random seeds, and shaded regions indicate 95% confidence intervals. Individual seed curves (except for FetchReach-v2) are exponentially smoothed with coefficient $\alpha = 0.9$ for visual clarity. Despite acting through a fixed, low-cardinality discrete action set, our method converges to a substantial fraction of the final returns obtained by the continuous-action PPO baseline on all tasks.

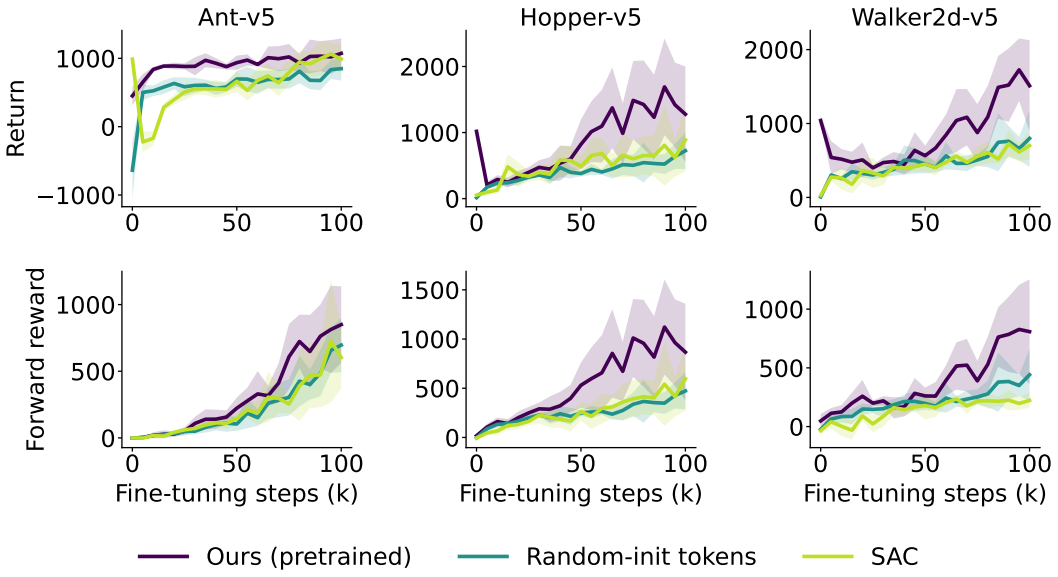

Figure 8: Downstream fine-tuning curves on Ant-v5, Hopper-v5, and Walker2d-v5. Curves show mean deterministic evaluation performance over $n = 10$ independent runs, and shaded regions indicate 95% confidence intervals. The top row reports standard return and the bottom row reports forward reward.

Under our fine-tuning objective ($\beta = 0.2$), token usage becomes more concentrated but does not collapse: approximately 28–32 effective tokens remain in use, the most-used token receives only 9–10% of the probability mass, and pairwise action distances remain close to their initial values. The observed action displacement shows that fine-tuning adapts the tokens to the downstream reward rather than leaving them frozen. Compared with random initialization, the pretrained policies retain substantially broader usage in all three environments: the random baseline ends with only 21 effective tokens on Ant-v5 and approximately 11 on Hopper-v5 and Walker2d-v5, with maximum usage

Table 5: Token-bank evolution during downstream fine-tuning. Arrows denote step $0 \rightarrow 100{,}000$. N_{eff} is the effective number of tokens under the marginal token-usage distribution; pairwise distance is the mean per-dimension RMS distance between executed token actions; maximum usage probability is the largest marginal token-usage probability; and action displacement is the mean per-dimension RMS distance between each current token action and the corresponding action produced by a frozen copy of the policy at fine-tuning step 0 on the same states.

Setting	N_{eff}	Pairwise dist.	Max. usage probability	Action disp.
Ant-v5, pretrained	$41.6 \rightarrow 32.0$	$0.50 \rightarrow 0.44$	$0.05 \rightarrow 0.09$	0.23
Ant-v5, random init.	$63.7 \rightarrow 21.1$	$0.95 \rightarrow 0.79$	$0.02 \rightarrow 0.12$	0.39
Ant-v5, pretrained, $\beta = 0$	$41.6 \rightarrow 1.0$	$0.50 \rightarrow 0.40$	$0.05 \rightarrow 1.00$	0.31
Hopper-v5, pretrained	$47.9 \rightarrow 27.7$	$0.56 \rightarrow 0.63$	$0.03 \rightarrow 0.10$	0.56
Hopper-v5, random init.	$63.7 \rightarrow 11.4$	$1.03 \rightarrow 0.88$	$0.02 \rightarrow 0.31$	0.60
Walker2d-v5, pretrained	$45.0 \rightarrow 31.7$	$0.54 \rightarrow 0.53$	$0.03 \rightarrow 0.09$	0.41
Walker2d-v5, random init.	$63.7 \rightarrow 10.5$	$0.98 \rightarrow 0.87$	$0.02 \rightarrow 0.29$	0.46

probabilities reaching 29–31% in the latter two environments. The pretrained policies also exhibit less action displacement in every environment.

We also fine-tune on Ant-v5 with the maxi-mix-mini-com term removed ($\beta = 0$). Although this setting achieves high standard-return AUC, its usage collapses to one token within 5,000 steps, while the component actions themselves remain distinct. Thus, the collapse concerns token usage: the fine-tuned mixture assigns essentially all marginal selection probability to one token, while the token actions remain distinct. This contrast shows that retaining the entropy objective during fine-tuning preserves broad use of the learned token bank.

Effect of token duration. Our objective learns distinct component actions at each state, with the downstream controller selecting among them at every environment step. Consistent with this one-step design, holding each selected token for $H = 5$ steps restricts the controller's ability to switch components as the state changes. On Ant-v5 with $K = 16$ and $\alpha = 0.2$, after 10^7 downstream environment steps, the mean episodic return across 10 random seeds was 2515.53 ± 308.08 for $H = 1$ and 1272.20 ± 101.20 for $H = 5$, with uncertainty reported as 95% confidence intervals.

Sample efficiency in a stabilization-constrained Ant variant. To further assess the practical benefits of our approach in settings where stabilization is genuinely challenging, we design a modified Ant-v5 environment that mirrors the "do not fall" constraint in a controlled way. Concretely, we constrain the torso height to a narrow healthy band $z \in [0.6, 0.7]$, compared to the standard MuJoCo band $[0.3, 1.0]$; episodes terminate immediately upon leaving this band, which substantially tightens the stabilization requirement (all episodes start at $z = 0.65$). The downstream reward is defined purely as forward velocity, with no survival bonus, to avoid giving our pretrained tokens an automatic advantage. We first pretrain a mixture policy using only the maxi–mix–mini–com objective for 500,000 interaction steps under this termination rule, then freeze the components and train a high-level controller for 200,000 steps on the forward-only task; PPO is trained from scratch for 200,000 steps on the same task. Our method achieves a return of 55.81 ± 6.62 with mean episode length 498.8 ± 102.67 steps, whereas PPO attains 5.32 ± 5.21 with length 7.4 ± 2.88. Thus, for the same downstream training budget, our pretrained tokens yield more than a tenfold improvement in average return over PPO and produce policies that survive for hundreds of steps. This behavior arises because maximizing entropy under the narrow healthy band forces the mixture components to maintain upright, stable dynamics over long horizons while spanning diverse actions within the safe region, whereas PPO trained from scratch remains in a near-immediate-fall regime with very short episodes and limited usable learning signal. Taken together with the Humanoid results, these findings indicate that, in environments where failure terminates behavior and stabilization is difficult, behavioral-token pretraining confers a tangible downstream sample-efficiency and robustness advantage on top of the compact, transferable discrete action interface established in our other experiments.

Effect of entropy regularization on component spacing. Subsequently, we tested the maxi-mix-mini-com entropy objective on the MuJoCo Ant benchmark (Todorov et al., 2012). 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

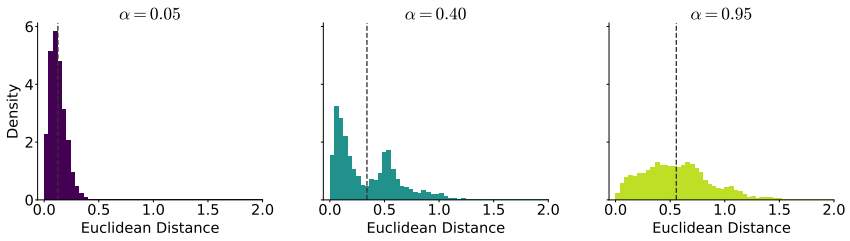

Figure 9: Pairwise Euclidean distance densities between component action means for $\alpha = 0.05, 0.4,$ and 0.95.

under deterministic execution. Figure 9 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 span of the action space and preserves small groups of highly similar, specialized components.

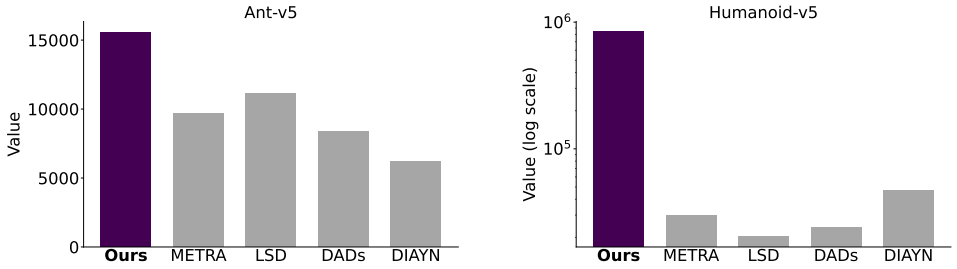

Figure 10: Within-mixture component divergence (maxi-mix-mini-com) vs. skill divergence (skill discovery methods), measured as the average sum of pairwise Jeffreys divergences per state over 20k evaluations (10k entropy-driven (Ramírez-Ruiz et al., 2024) + 10k uniform) on Ant-v5 and Humanoid-v5; ours shown with $\alpha = 0.8$ (Ant) and $\alpha = 1.0$ (Humanoid).

Measuring within-mixture diversity. We seek to quantify how diverse our behavioral tokens are compared to the diversity between skills learned by skill discovery methods, since greater separation yields a richer bank of behavioral tokens (quantized actions) for composition. Concretely, for our approach we compute, at each state, the sum of pairwise Jeffreys (symmetrized KL) divergences between the components $\{\pi_k(\cdot|s)\}$ of the same mixture policy $\pi_m(a|s)$; for each skill discovery method, we analogously compute divergences between its skills $\pi(\cdot|s, z)$ across latent codes z. We average this per-state quantity over $20{,}000$ evaluation states assembled from two exploration regimes: (i) an entropy-driven regime using MOP (Ramírez-Ruiz et al., 2024), where MOP is trained to maximize the agent's policy entropy for 1×10^6 environment steps and then executed to collect trajectories until $10k$ states are obtained; and (ii) a uniform-action regime in which actions are sampled uniformly at random from the environment action space and executed to collect $10k$ states (no training). We report the resulting means on MuJoCo Ant-v5 and Humanoid-v5 (Fig. 10). In the configurations shown (α=0.8 for Ant; α=1.0 for Humanoid), our mixture attains the largest average pairwise Jeffreys divergence (Fig. 10), indicating more distinct behavioral tokens within the mixture. Supplementary videos (see Videos 5) illustrate that the mixture components exhibit greater behavioral diversity and yield a broader behavioral repertoire with more effective actions.

In addition, we report the mean pairwise Euclidean distance (MPED) for various combinations of α and K in Ant-v5, averaged over 5 random seeds (Table 6). The results indicate that both increasing α and K increasing lead to higher MPED values, reflecting greater diversity among components. Notably, the effect of α is more pronounced when K is small, as components must spread more aggressively to achieve higher overall entropy. In contrast, for larger K, diversity is already induced by the increased number of components, each of which must occupy a distinct region of the action space to maintain low entropy while contributing to the overall mixture entropy.

Table 6: Mean pairwise Euclidean distance for various α and K values for Ant-v5.

K	$\alpha = 0.10$	$\alpha = 0.20$	$\alpha = 0.30$
16	0.58	0.96	1.32
64	1.24	1.30	1.33
128	1.33	1.34	1.35

Table 7: Performance for Humanoid, Walker2D, FetchReach and Swimmer under different action-set configurations with $K = 4$, 16, and 64 components. Values are mean $\pm 95\%$ CI. "m" uses the discrete alphabet $\{-m, 0, +m\}$ to sample each action dimension independently.

Env.	Method	$K = 4$	$K = 16$	$K = 64$
FetchReach-v2	PPO		0.80 ± 0.18	
	Ours	0.460 ± 0.04	1.000 ± 0.000	1.000 ± 0.000
	$\mathcal{U}(-1.0, 1.0)$	0.16 ± 0.09	0.96 ± 0.02	1.000 ± 0.000
	$\mathcal{N}(\mu = 0, \sigma = 0.2)$	0.14 ± 0.09	0.76 ± 0.06	0.88 ± 0.04
	$m = 0.1$	0.180 ± 0.050	0.360 ± 0.051	0.140 ± 0.060
	$m = 0.2$	0.080 ± 0.006	0.540 ± 0.125	0.620 ± 0.139
	$m = 0.5$	0.320 ± 0.013	0.940 ± 0.060	1.000 ± 0.000
Swimmer	PPO		98.07 ± 6.58	
	Ours	89.41 ± 2.11	109.96 ± 3.80	119.05 ± 3.03
	$\mathcal{U}(-1.0, 1.0)$	77.50 ± 1.31	103.87 ± 10.78	111.33 ± 7.54
	$\mathcal{N}(\mu = 0, \sigma = 0.2)$	44.36 ± 2.28	86.73 ± 3.79	42.60 ± 3.35
	$m = 0.1$	34.99 ± 4.22	46.50 ± 2.12	48.52 ± 2.10
	$m = 0.2$	47.25 ± 2.84	45.23 ± 6.64	54.52 ± 2.60
	$m = 0.5$	30.56 ± 4.72	83.59 ± 5.05	76.08 ± 3.86
Walker2D	PPO		3834.31 ± 160.21	
	Ours	1479.30 ± 44.76	2394.14 ± 63.61	2473.93 ± 234.04
	$\mathcal{U}(-1.0, 1.0)$	11.52 ± 0.87	2031.73 ± 367.35	1434.98 ± 296.94
	$\mathcal{N}(\mu = 0, \sigma = 0.2)$	351.02 ± 42.17	369.71 ± 048.80	486.66 ± 39.71
	$m = 0.1$	289.02 ± 53.23	357.84 ± 30.44	466.15 ± 40.78
	$m = 0.2$	296.34 ± 69.01	748.64 ± 120.44	485.64 ± 40.44
	$m = 0.5$	1016.90 ± 250.58	447.81 ± 78.24	1995.08 ± 57.28
	$m = 1.0$	438.46 ± 28.89	1676.77 ± 35.62	749.45 ± 55.59
Humanoid	PPO		2285.35 ± 948.65	
	Ours	5092.87 ± 42.45	5360.89 ± 108.05	5672.19 ± 62.49
	$\mathcal{U}(-0.4, 0.4)$	525.78 ± 62.45	512.46 ± 41.04	610.24 ± 51.57
	$\mathcal{N}(\mu = 0, \sigma = 0.2)$	418.04 ± 11.74	522.18 ± 07.14	508.72 ± 41.10
	$m = 0.1$	487.29 ± 29.88	467.53 ± 28.71	544.66 ± 30.21
	$m = 0.2$	561.22 ± 33.12	608.85 ± 10.29	583.90 ± 20.21
	$m = 0.4$	415.37 ± 18.92	489.64 ± 22.15	503.28 ± 15.47

How sensitive is performance to the number of tokens and specialization strength? Because K sets the downstream interface size and α controls specialization, we examine them jointly on Ant-v5. Averaged across the tested α values, mean return increases by 43.1% from $K = 16$ to $K = 64$, compared with 15.2% from $K = 64$ to $K = 128$ (Table 8). The best α is 0.2 for $K = 16$ and $K = 64$, and 0.1 for $K = 128$. Pretraining runtime remains between 3.4 and 3.6 Ksec for 200K interactions across all three values of K, averaged over 5 seeds.

Table 8: Ant-v5: returns after 10M steps. Values are mean $\pm 95\%$ CI. "Uniform" draws each action dimension from $[-1, 1]$. "m" uses the discrete alphabet $\{-m, 0, +m\}$ to sample each action dimension independently.

Method	Parameter	16	64	128
PPO	–		3176.44 ± 224.21	
Ours	$\alpha = 0.1$	1669.40 ± 22.22	2399.07 ± 18.95	3726.89 ± 32.66
	$\alpha = 0.2$	2936.96 ± 23.18	3580.79 ± 31.87	3399.25 ± 20.05
	$\alpha = 0.3$	1864.21 ± 17.72	3280.89 ± 23.05	3538.76 ± 31.33
Random Set of Actions	$m = 1.0$	2677.58 ± 281.31	1287.32 ± 435.12	1516.34 ± 125.42
	$m = 0.5$	2339.68 ± 202.66	3198.97 ± 397.03	1783.20 ± 096.99
	$m = 0.2$	1883.51 ± 264.06	1751.37 ± 255.22	1769.44 ± 292.19
	$m = 0.1$	1044.44 ± 081.46	1059.06 ± 156.44	0955.92 ± 174.86
	$\mathcal{U}(-1.0, 1.0)$	2698.80 ± 372.46	1541.63 ± 196.43	1101.21 ± 206.97
	$\mathcal{N}(\mu = 0, \sigma = 0.2)$	1841.58 ± 060.78	2470.82 ± 120.45	2335.90 ± 180.36

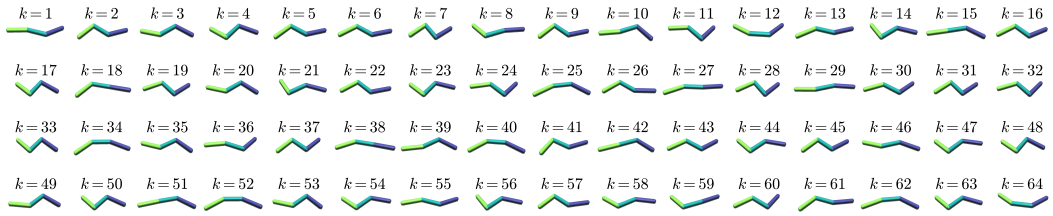

Figure 11: MuJoCo Swimmer-v5 poses resulting from continuously applying the mode of each learned component of a 64-component Gaussian mixture policy as the action. These results illustrate the diversity and coverage of the continuous action space.

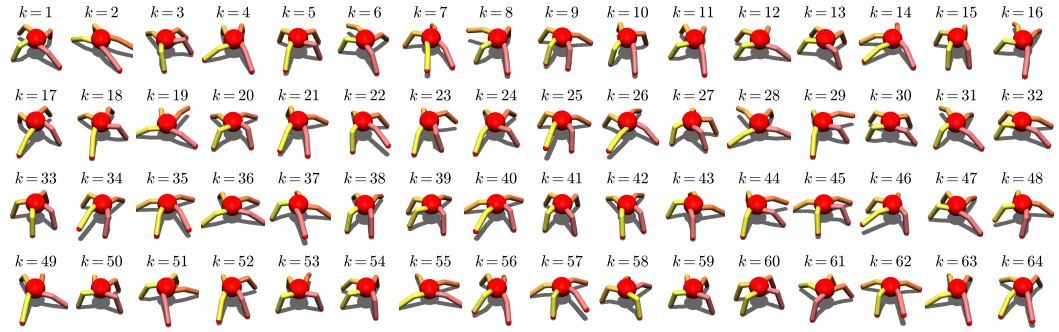

Figure 12: MuJoCo Ant-v5 poses resulting from continuously applying the mode of each learned component of a 64-component Gaussian mixture policy as the action, showing diversity and coverage of the continuous action space.

Table 9: Hyperparameters

Parameter	Value
Optimizer	Adam (Kingma & Ba, 2015)
Learning rate	3×10^{-4}
Discount (γ)	0.9
Replay buffer size	10^5
Shared encoder and critic	2 layers, 256 units each
Component network	2 layers, 32 units each
Number of samples per minibatch	32
Steps per epoch	2000
Initial random steps	2000
Nonlinearity	ReLU
Target network smoothing coefficient (τ)	0.005 (Polyak & Juditsky, 1992)

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 $\{V^{(n)}, \pi^{(n)}, w^{(n)}, r^{(n)}\}_{n=0}^{N}$ with $V^{(n)}(s)$ non-decreasing for every state s..

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

Theorems 3 and 5 are exact on-policy identities under $d_{\pi_{m,\theta}}$. Algorithm 2 instead samples states from the replay buffer, learns two critics Q_{ϕ_j} using target critics, and uses a straight-through estimator for the mixture weights, so its gradient update is an off-policy approximation to these identities. Replay minibatches sample states from experience collected under earlier policies, while each actor update samples a fresh action from the current mixture at those states. These replay-weighted updates are not an unbiased estimate of the on-policy theorem's gradient. Each critic $Q_{\phi_j}(s,a)$ estimates the discounted continuation value $\gamma Q(s,a)$, where Q is defined in Eq. 3, so the displayed target y_i is its one-step Bellman target for the same objective.

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 $\mathcal{D}$
Define $\pi_{k,\theta}$ by $\log \sigma_k \in [-20, 2]$, $z = \mu_k(s) + \sigma_k(s)\epsilon$, and $a = \tanh(z)$, where $\epsilon \sim \mathcal{N}(0, I)$
Evaluate $\log \pi_{m,\theta}(a|s) = \text{logsumexp}_k\big(\log w_{k,\theta}(s) + \log \pi_{k,\theta}(a|s)\big)$ using tanh-corrected component densities
for each iteration **do**
  for each environment step **do**
    Sample component index: $k_t \sim \text{GumbelSoftmax}\big(w_\theta(s_t), \tau\big)$
    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: $k_t' \sim \text{GumbelSoftmax}\big(w_\theta(s_i'), \tau\big)$
      Sample next action: $a_i' \sim \pi_{k_i',\theta}(\cdot|s_i')$
      Compute target:

$$y_i = \gamma(1 - d_i)\Big[\min_{j\in\{1,2\}} Q_{\phi_j}^{\text{targ}}(s_i', a_i') - \log \pi_{m,\theta}(a_i'|s_i') + \alpha \log \pi_{k_i',\theta}(a_i'|s_i')\Big]$$

    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:

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

    Compute the policy gradient:

$$\nabla_\theta J(\pi_\theta^m) \approx \frac{1}{B}\sum_{i=1}^{B} \nabla_\theta\Big[\min_{j\in\{1,2\}} Q_{\phi_j}(s_i, \tilde{a}_i) - \log \pi_{m,\theta}(\tilde{a}_i|s_i) + \alpha \log \pi_{k_i,\theta}(\tilde{a}_i|s_i)\Big]$$

    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