# A Mixture of Surprises for Unsupervised Reinforcement Learning

**Andrew Zhao** [1*] **Matthieu Gaetan Lin** [2*] **Yangguang Li**[3] **Yong-Jin Liu**[2†]

**Gao Huang** [1†]

[1] Department of Automation, BNRist, Tsinghua University
[2] Department of Computer Science, BNRist, Tsinghua University
[3] SenseTime
{zqc21,lyh21}@mails.tsinghua.edu.cn,
liyangguang@sensetime.com
{liuyongjin,gaohuang}@tsinghua.edu.cn,

## Abstract

Unsupervised reinforcement learning aims at learning a generalist policy in a reward-free manner for fast adaptation to downstream tasks. So far, existing methods propose to provide an intrinsic reward based on surprise. Maximizing or minimizing surprise drives the agent to either explore or gain control over its environment. However, both strategies rely on a strong assumption: the entropy of the environment's dynamics is either high or low. This assumption may not always hold in real-world scenarios, where the entropy of the environment's dynamics may be unknown. Hence, choosing between the two objectives is a dilemma. We propose a novel yet simple mixture of policies to address this concern, allowing us to optimize an objective that simultaneously maximizes and minimizes the surprise. Concretely, we train one mixture component whose objective is to maximize the surprise and another whose objective is to minimize the surprise. Hence, our method does not make assumptions about the entropy of the environment's dynamics. We call our method a **M**ixture **O**f **S**urprise**S** (MOSS) for unsupervised reinforcement learning. Experimental results show that, surprisingly, our simple method achieves state-of-the-art performance on the URLB benchmark, outperforming previous pure surprise maximization-based objectives. Our code will be made publicly available.

## 1 Introduction

Humans can learn meaningful behaviors without external supervision, i.e., in an unsupervised manner, and then adapt those behaviors to new tasks [3]. Inspired by this, unsupervised reinforcement learning decomposes the reinforcement learning (RL) problem into a pretraining phase and a finetune phase [32]. During a pretraining phase, an agent prepares all possible tasks that a user might select. Afterward, the agent tries to figure out the selected task as quickly as possible during finetuning [24]. Doing so allows solving the RL problem in a meaningful order [3], e.g., a cook first has to look at what is in the fridge before deciding what to cook. Unsupervised representation learning has shown great success in Computer Vision [27] and Natural Language Processing [10]; however, one challenge is that RL includes both behavior learning and representation learning [32, 53].

[*] Equal contribution.
[†] Corresponding authors.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

Current unsupervised RL methods provide an intrinsic reward to the agent during a pretraining phase to tackle the behavior learning problem [32]. Intuitively, this intrinsic reward should incentivize the agent to understand its environment [15]. Current methods formulate the intrinsic reward as either maximizing or minimizing surprise [45, 6, 36, 57, 42, 43, 11, 35, 31, 17, 47, 46, 13], where knowledge-based methods quantify surprise as the uncertainty of a prediction model [11, 42, 43] and data-based methods measure surprise as an information-theoretic quantity [45, 6, 57, 36, 5][3]. Surprise maximization methods [24, 31, 36] formulate the problem as an exploration problem. Intuitively, to prepare all possible downstream tasks, an agent has to explore the state space and figure out what is possible in the environment. For instance, one instantiation is to maximize the agent's state entropy [36]. In contrast to surprise maximization, another line of work takes inspiration from the free-energy principle [22, 21, 20] and proposes to minimize the surprise [6, 45]. In particular, these works argue that external perturbations naturally provide the agent with surprising events. Hence, an agent can learn meaningful behaviors by minimizing its surprise, and minimizing this quantity requires gaining control over these external perturbations [45, 6]. For example, SMiRL [6] proposed to minimize the agent's state entropy.

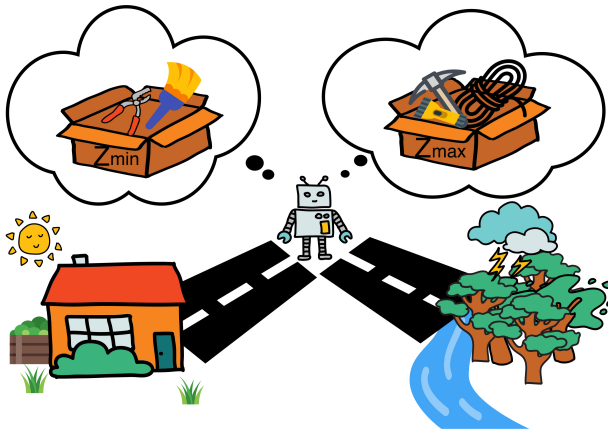



Figure 1: **Outline of our Mixture Of SurpriseS (MOSS) strategy**. We provide the agent with two paths, each corresponding to maximizing or minimizing surprise. During the pretraining phase, the agent gathers experience from both paths using the skills from the corresponding toolbox. Finally, during the finetuning phase, the agent can choose any skills from any toolbox.

A closer look at these two approaches indicates a strong assumption. Substantial external perturbations already naturally provide an agent with a high state entropy [6]. Therefore, surprise-seeking approaches assume that the environment does not provide significant external perturbations. On the other hand, in an environment without any external perturbations, the agent already achieves minimum entropy by not acting [45]. Therefore, surprise minimization approaches assume that the environment provides external perturbations for the agent to control. However, in real-world scenarios, it is often difficult to quantify the entropy of the environment's dynamics beforehand, or the agent might face both settings. For example, a butler robot performs mundane chores daily in a household. However, the robot might also encounter surprising events (e.g., putting out a fire). Therefore a fixed assumption on the entropy of the environment dynamics is often not possible. In other words, choosing between surprise maximization or minimization for pretraining is a dilemma.

Simultaneously optimizing these two opposite objectives does not make sense. Instead, in this paper, we show that competence-based methods [32] offer a simple yet effective way to combine the benefits of these two objectives. In addition to conditioning the policy on the state, competence-based methods condition the policy on a latent vector [31, 35, 26, 55, 23, 1, 18]. Conditioning the policy offers an appealing way to formulate the unsupervised RL problem. During a pretraining phase, the agent tries to learn a set of possible behaviors in the environment and distills them into skills. Then, during finetuning, the agent tries to figure out the selected task as quickly as possible, using the repertoire of skills gathered during pretraining. For example, as illustrated in Fig. 1, we view skill distributions as a mixture of policies instead of a single policy. In particular, we train one set of skills whose objective is to maximize the surprise and another set of skills whose objective is to

[3]Refer to [32] for detailed definitions of data-based, knowledge-based and competence-based methods.

minimize the surprise. Surprisingly, this paper shows that this simple approach works well in practice, which simultaneously optimizes two contradicting objectives. Our primary contribution, presented in Section 4, is a simple intrinsic reward called MOSS that does not make assumptions about the entropy of the environment's dynamics. In Section 5, our experimental results on URLB [32] and ViZDoom [30] show that, surprisingly, our MOSS methods achieve state-of-the-art results.

We organize the paper as follows. Section 3 briefly analyzes previous unsupervised RL algorithms under the surprise framework. Then in Section 4, we introduce our MOSS method. Next, experimental results in Section 5 shows that on URLB [32] and ViZDoom [30], our MOSS method improves upon previous pure maximization and minimization methods. Finally, we provide discussions and limitations in Section 6.

## 2 Preliminaries

**Markov Decision Process** Unsupervised RL methods studied in this paper operate under a Markov Decision Process (MDP) [51]. In particular, we specify an MDP as a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, T, r^{\text{ext}}, \rho, \gamma)$ where $\mathcal{S}$ is the state space and $\mathcal{A}$ is the action space of the environment. $T(\mathbf{S}' \mid \mathbf{S}, \mathbf{A})$ represents the state-transition dynamics, $\rho : \mathcal{S} \to [0, 1]$ is the initial state distribution, and $\gamma \in [0, 1)$ is the discount factor. At each discrete time step $t \in \mathbb{Z}^*$, the agent receives a state and performs an action, which we denote as $\mathbf{S}_t \in \mathcal{S}$ and $\mathbf{A}_t \in \mathcal{A}$, respectively. During pretraining, unsupervised RL algorithms compute an intrinsic reward $r^{\text{int}}$; during the finetune phase, the agent receives the extrinsic reward $r^{\text{ext}}$ given by the environment at each interaction.

**Skill** Intuitively, a *skill* is an abstraction of a specific behavior (e.g., walking), and in practice, a *skill* is a latent conditioned policy [17]. Given a latent vector $\mathbf{z}$, we denote a skill as $\pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t, \mathbf{z})$, where $\pi_\theta$ is the policy parameterized by $\theta$. For instance, during pretraining, the latent vectors are sampled every $n$ steps such that the latent vector $\mathbf{z}$ is associated with the behavior executed during the associated $n$ steps.

**Mutual Information** Knowledge-based, data-based, and competence-based methods have different measures of surprise. The study in this paper falls into the category of competence-based methods. In particular, data-based and competence-based methods rely on an information-theoretic definition of surprise, i.e., entropy. Previous competence-based methods acquire skills by maximizing mutual information [16] between $\mathcal{T}$ and skills $\mathbf{Z}$

$$\mathbb{I}(\mathcal{T}; \mathbf{Z}) = \mathbb{H}[\mathcal{T}] - \mathbb{H}[\mathcal{T}|\mathbf{Z}] \tag{1}$$

$$= \mathbb{H}[\mathbf{Z}] - \mathbb{H}[\mathbf{Z}|\mathcal{T}], \tag{2}$$

where $\mathcal{T}$ can be the states $\mathbf{S}$, the joint distribution of state-transitions $(\mathbf{S}', \mathbf{S})$, or the state-transitions $(\mathbf{S}' \mid \mathbf{S})$. In particular, these methods differ in how they decompose the mutual information. Theoretically, these different decompositions are equivalent, i.e., they all maximize the mutual information between states and skills. However, the particular choice greatly influences the performance in practice as optimizing this objective relies on approximations.

To motivate the potential of competence-based methods over data-based or knowledge-based methods, we provide an intuitive understanding of Eq. 1. On the one hand, the entropy term says that we want skills in aggregate that explore the state space; we use it as a proxy to learn skills that cover the set of possible behaviors. On the other hand, It is not enough to learn skills that randomly go to different places. We want to reuse those skills as accurately as possible, meaning we need to be able to discriminate or predict the agent's state transitions from skills. To do so, we minimize conditional entropy. In other words, appropriate skills should cover the set of possible behaviors and should be easily distinguishable.

## 3 Information-Theoretic Skill Discovery

Competence-based methods employ different intrinsic rewards to maximize mutual information: (1) discriminability-based and (2) exploratory-based intrinsic rewards. For example, the former rewards the agent for discriminable skills. In contrast, the latter rewards the agent for skills that effectively cover the state space using a KNN density estimator [48, 36] to approximate the entropy term. Below we analyze both approaches.

### 3.1 Discriminability-based Intrinsic Reward

Previous work such as DIAYN [17] uses a discriminability-based intrinsic reward. They use the decomposition in Eq. 2 and given a variational distribution $q_\phi$, they optimize the following variational lower bound

$$\begin{aligned}\mathbb{I}(\mathbf{S};\mathbf{Z}) &= \mathbb{KL}(p(\mathbf{s},\mathbf{z}) \mid\mid p(\mathbf{s})p(\mathbf{z})) \\ &= \mathbb{E}_{\mathbf{s},\mathbf{z}\sim p(\mathbf{s},\mathbf{z})}\left[\log \frac{q_\phi(\mathbf{z} \mid \mathbf{s})}{p(\mathbf{z})}\right] + \mathbb{E}_{\mathbf{s}\sim p(\mathbf{s})}[\mathbb{KL}(p(\mathbf{z} \mid \mathbf{s}) \mid\mid q_\phi(\mathbf{z} \mid \mathbf{s}))] \\ &\geq \mathbb{E}_{\mathbf{z},\mathbf{s}\sim p(\mathbf{z},\mathbf{s})}[\log q_\phi(\mathbf{z} \mid \mathbf{s})] - \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}[p(\mathbf{z})], \end{aligned} \tag{3}$$

where $\mathbf{Z} \sim p(\mathbf{Z})$ is a discrete random variable. In particular, the discriminator rewards the agent if it can guess the skill from the state, i.e., $r^{\text{int}}(\mathbf{s}) \triangleq \log q_\phi(\mathbf{z} \mid \mathbf{s}) - \log p(\mathbf{z})$. These methods may easily run into a chicken and egg problem, where skills learn to be diverse using the discriminator's output. However, the discriminator cannot learn to discriminate skills if the skills are not diverse. Hence, it discourages the agent from exploring. Previous work [49] has tried to solve this problem by decoupling the *aleatoric uncertainty* from the *epistemic uncertainty*.

Furthermore, solving the chicken and egg problem is not enough since a state must map to a single skill in Eq. 2. Accordingly, the methods that use the decomposition in Eq. 2 require the skill space to be smaller than the state space so that the skills are distinguishable. Since previous work showed that a high-dimensional continuous skill space empirically performs better, the intrinsic reward should not rely on the decomposition in Eq. 2. Intuitively, a continuous skill space allows (1) to interpolate between skills and (2) to represent a more significant set of skills in a more compact representation.

Instead, in Eq. 1, in contrast to skills that are predictable by states $\mathbb{H}[\mathbf{Z} \mid \mathcal{T}]$, we require that states are predictable by skills $\mathbb{H}[\mathcal{T} \mid \mathbf{Z}]$. Since any state should be predictable from a given skill, the skill space must be larger than the state space (i.e., we do not want a skill mapping to multiple states). Therefore, another work [47] relies on a variational bound on Eq. 1.

$$\begin{aligned}\mathbb{I}(\mathbf{S}';\mathbf{Z} \mid \mathbf{S}) &= \mathbb{E}_{\mathbf{z},\mathbf{s},\mathbf{s}'\sim p(\mathbf{z},\mathbf{s},\mathbf{s}')}\left[\log \frac{q_\phi(\mathbf{s}' \mid \mathbf{s},\mathbf{z})}{p(\mathbf{s}' \mid \mathbf{s})}\right] + \mathbb{E}_{\mathbf{z},\mathbf{s}\sim p(\mathbf{z},\mathbf{s})}\big[\mathbb{KL}(p(\mathbf{s}' \mid \mathbf{s},\mathbf{z}) \mid\mid q_\phi(\mathbf{s}' \mid \mathbf{s},\mathbf{z}))\big] \\ &\geq \mathbb{E}_{\mathbf{z},\mathbf{s},\mathbf{s}'\sim p(\mathbf{z},\mathbf{s},\mathbf{s}')}\left[\log \frac{q_\phi(\mathbf{s}' \mid \mathbf{s},\mathbf{z})}{p(\mathbf{s}' \mid \mathbf{s})}\right], \end{aligned} \tag{4}$$

which uses a continuous skill space; however, it does not scale to high dimensions as the intrinsic reward $r^{\text{int}}(\mathbf{s}',\mathbf{a},\mathbf{s}) \triangleq \log \frac{q_\phi(\mathbf{s}'|\mathbf{s},\mathbf{z})}{p(\mathbf{s}'|\mathbf{s})}$ relies on an estimation of $p(\mathbf{s}' \mid \mathbf{s})$. In particular, they assume that $p(\mathbf{z} \mid \mathbf{s}) = p(\mathbf{z})$. Intuitively, given $\mathbf{s}$, if we assume each element in the latent vector $(\mathbf{z})_i$ to be independent of each other, the error of this assumption will be scaled by the dimension of $\mathbf{z}$.

### 3.2 Exploratory-based Intrinsic Reward

As aforementioned, discriminability-based intrinsic reward may easily run into a chicken and egg problem. Hence, other work [31, 35] uses an exploratory-based intrinsic reward to address the chicken and egg problem. In other words, they use the decomposition in Eq. 1, where the intrinsic reward explicitly rewards the agent for exploring through $\mathbb{H}[\mathcal{T}]$.

Previous work [36] maximizes the state entropy, i.e., $\mathcal{T} = \mathbf{S}$. However, the authors in [4] argue that it often results in the discriminator simply memorizing the last state of each skill. Instead, CIC [31] proposes to maximize the entropy of the joint distribution of state-transitions (which from now on we refer as joint entropy), i.e., $\mathbb{H}[\mathbf{S}',\mathbf{S}]$.

Therefore, CIC [31] proposes to estimate the conditional entropy $\mathbb{H}[\mathbf{S}',\mathbf{S} \mid \mathbf{Z}]$ using noise contrastive estimation [31, 25] and the joint entropy $\mathbb{H}[\mathbf{S}',\mathbf{S}]$ using a $k$-nearest neighbor estimator [48] to handle high dimensional skill space.

**KNN-density estimation** Previous works [36, 31] approximate the joint entropy $\mathbb{H}[\mathbf{S}',\mathbf{S}]$ using a $k$-nearest neighbor estimator [48]. Given a random sample of size $N$, $\{(\mathbf{S}^{(i)'},\mathbf{S}^{(i)})\}_{i=1}^{N} \sim P(\mathbf{S}',\mathbf{S})$,

the estimator computes a Monte Carlo estimate [7] of the entropy defined as

$$\hat{\mathbb{H}}[\mathbf{S}', \mathbf{S}] = -\frac{1}{N}\sum_{i=1}^{N} \log \hat{p}_k(\mathbf{s}^{(i)'}, \mathbf{s}^{(i)}) + b(k), \tag{5}$$

where $\hat{p}_k(\mathbf{s}^{(i)'}, \mathbf{s}^{(i)}) = \frac{k}{NV_k^{(i)}}$ is an estimation of $p(\mathbf{s}^{(i)'}, \mathbf{s}^{(i)})$, $V_k^{(i)}$ is the volume of the hypersphere of radius $R_k^{(i)}$, and $b(k)$ is a constant such that the estimator is unbiased. In particular, $R_k$ is the distance from $\mathbf{s}$ to its $k$-nearest neighbor $\mathbf{s}'_k$ where we assume a uniform distribution of points within the hypersphere (local uniformity assumption [37]).

In practice, to make the distance $R_k$ meaningful, we do not operate in the state space. For instance, CIC [31] trains an MLP $f_\psi : \mathbb{R}^{|\mathcal{S}|} \to \mathbb{R}^d$ that projects the state into a latent representation such that $R_k = \|f_\psi(s) - f_\psi(s'_k)\|$ where $d$ is the skill dimension. Moreover, in APT [36], it was found that averaging over all $k$-nearest neighbors leads to better results. Hence, CIC [31] optimizes a quantity proportional to Eq. 5:

$$\hat{\mathbb{H}}[\mathbf{S}', \mathbf{S}] \triangleq \sum_{i=1}^{N} \log\left(c + \frac{1}{k}\sum_{k} R_k^{(i)}\right), \tag{6}$$

where $c = 1$ is constant for numerical stability. We view the joint entropy as an expected reward, and for a transition $(\mathbf{s}^{(i)}, \mathbf{s}^{(i)'})$, the reward function is

$$r^{\text{int}}(\mathbf{s}^{(i)}, \mathbf{s}^{(i)'}) \triangleq \log\left(c + \frac{1}{k}\sum_{k} R_k^{(i)}\right). \tag{7}$$

**Noise Contrastive Estimation** To train parameters $\psi$, CIC [31] minimizes the conditional entropy $\mathbb{H}[\mathbf{S}'; \mathbf{S} \mid \mathbf{Z}]$ using a noise contrastive estimation (NCE) [25, 40]. Denote $\mathbf{h} = f_\psi(\mathbf{s})$ and $\mathbf{h}' = f_\psi(\mathbf{s}')$ the projected representations of $\mathbf{s}$ and $\mathbf{s}'$, respectively. Furthermore, let $g_{\phi_z} : \mathbb{R}^d \to \mathbb{R}^d$ be an MLP that projects the skill into a latent representation and $g_{\phi_s} : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^d$ be an MLP that projects $\mathbf{h}$ and $\mathbf{h}'$ into a latent representation. We define the NCE loss

$$\mathcal{L}_{\text{nce}}(\mathbf{s}^{(i)}, \mathbf{z}^{(i)}) = -\log \frac{\exp c_i^{(i)}}{\sum_{j=1}^{N} \exp c_j^{(i)}} \tag{8}$$

as an approximation to $\mathbb{H}[S'; S \mid Z]$, where we define $\cos(\cdot, \cdot)$ as the cosine similarity operator, $\omega$ as a temperature term, and $c_j^{(i)} = \cos(g_{\phi_z}(\mathbf{z}^{(i)}), g_{\phi_s}(\mathbf{h}^{(j)}, \mathbf{h}^{(j)'}))/\omega$. As aforementioned, this term serves as representation learning for parameters $\psi$. Intuitively, the NCE loss [25] pushes $p(\mathbf{s}' \mid \mathbf{s}, \mathbf{z})$ to be a delta-like density function. However, since we define behaviors with $n$ steps, we want a wider distribution, so in practice, $\omega$ is set to $0.5$ to smooth the distribution.

## 4 Mixture Of Suprises (MOSS)

Our goal is to provide an algorithm that does not build on any assumption about the entropy of the environment's dynamics. A straightforward way to achieve this is to maximize and minimize surprise simultaneously. A benefit of such an objective is that it promotes exploration to find a more effective policy for minimizing the surprise [6]. In particular, in [6], they maximize surprise according to all prior experiences while minimizing surprise according to the current episode. However, they find that it only helps in some cases. The challenge is to find an effective way to optimize these two opposite objectives since if we add the two objectives, it would simply yield a linear interpolation of the two objectives, which still corresponds to either maximizing *or* minimizing. A previous work [19], dubbed Adversarial Surprise (AS), resolves this challenge using an adversarial game where two policies compete against each other: one maximizes surprise while the other minimizes surprise. AS [19] requires training two policies that do not share parameters or data. Instead, we propose a novel yet simple mixture of policies to maximize and minimize surprise simultaneously. Our mixture of policies does not rely on an adversarial game [19] in which training is known to be challenging [38] and uses a single network for both objectives resulting in higher sample efficiency. Mainly, our method only

**Algorithm 1** MOSS: Unsupervised pretraining

**Input:** Environment, number of pretraining steps $N_T$.
Initialize DDPG, and $\phi_s, \phi_z, \psi$
**for** $t = 1$ **to** $N_T$ **do**
  **if** ($t$%update skill every) $== 0$ **then**
    Sample $m_t \sim p(m)$ ▷ setting to $m = 0$ is equivalent to CIC[31]
    Sample $\mathbf{z}_t \sim p(\mathbf{z} \mid m_t)$
  **end if**
  Take some action $\mathbf{a}_t$ and observe $\mathbf{s}_{t+1}$
  Store $(\mathbf{s}_t, \mathbf{s}_{t+1}, m_t, \mathbf{z}_t)$ into buffer $\mathcal{B}$
  **if** $t \geq 4000$ **then**
    Sample a batch $\{(\mathbf{a}^{(i)}, \mathbf{s}^{(i)}, \mathbf{s}^{(i)'}, m^{(i)}, \mathbf{z}^{(i)})\}_{i=1}^{N}$ and compute intrinsic reward $r^{\text{int}}$ using equation 9.
    Update DDPG using intrinsic reward.
    Update $\phi_s, \phi_z, \psi$ using noise contrastive loss in equation 8.
  **end if**
**end for**

requires a minor change to CIC [31] that does not involve any additional computational cost during training. In particular, our mixture of policies builds on a competence-based approach [31], meaning we use two disjoint skill sets. We separate the two opposite objectives using a different skill set for each objective: one skill set for surprise maximization and another for surprise minimization.

Specifically, we define the mixture of policies using a binary random variable $M$, where we associate each mixture component with a continuous uniform distribution $\mathcal{U}^d(a, b)$ on an interval $[a, b]$. Each uniform distribution constitutes a skill distribution. Maintaining two different skill distributions allows us to define a different objective for each distribution. In particular, we define two distributions $\mathbf{Z}_{\max} \sim P(\mathbf{Z} \mid M = 0)$ and $\mathbf{Z}_{\min} \sim P(\mathbf{Z} \mid M = 1)$. Since we do not assume any priors on the entropy of the environment's dynamics, we deterministically set $M = 0$ for the first half of the steps and $M = 1$ for the other half of the steps in the episode. Compared to more sophisticated methods (e.g., appendix B) for setting $M$ (i.e., switching the objective), such a heuristic adds no computational cost during training, requires no hyper-parameter tuning, and performs well.

Moreover, motivated by the discussions in Section 3.2, we build our method on top of the CIC [4] [31] framework. This framework is appealing as it bypasses the chicken and egg problem and supports high-dimensional skills. In addition, it significantly outperforms previous competence-based methods on the URLB benchmark [32]. Hence, following CIC [31], we use the decomposition in Eq. 1

$$\mathbb{I}(\mathbf{S}', \mathbf{S}; \mathbf{Z}) = \mathbb{H}[\mathbf{S}', \mathbf{S}] - \mathbb{H}[\mathbf{S}', \mathbf{S} \mid \mathbf{Z}].$$

In particular, we define *surprise* as the joint entropy, which corresponds to maximizing and minimizing $\mathbb{H}[\mathbf{S}', \mathbf{S}]$, and as in CIC [31] we use Eq. 6 to approximate the joint entropy. We approximate the conditional entropy $\mathbb{H}[\mathbf{S}', \mathbf{S} \mid \mathbf{Z}]$ using a noise contrastive estimation following Eq. 8. By minimizing the conditional entropy, we distill behaviors into skills. Doing so also serves as representation learning for computing the joint entropy [31].

**Implementation** To ensure fairness, all hyper-parameters are kept as in CIC [31] and we use the same RL algorithm (i.e., DDPG [34] as implemented in DrQ-V2 [56]).

In practice, $\mathbf{Z}_{\max} \sim P(\mathbf{Z} \mid M = 0) \triangleq \mathcal{U}^d(0, 1)$ and $\mathbf{Z}_{\min} \sim P(\mathbf{Z} \mid M = 1) \triangleq \mathcal{U}^d(-1, 0)$ (we refer readers to appendix for other variants that we tried), where we set $M = 0$ for the first half of steps and $M = 1$ for the remaining half of steps in the episode. Hence, the RL agent maximizes

$$\mathcal{F}(\theta) \triangleq -\mathbb{E}_{\mathbf{z}_{\max} \sim \mathcal{U}^d(0,1)} \mathbb{E}_{p_{\mathbf{z}_{\max}}(\mathbf{s}', \mathbf{s})} [\log p_{\mathbf{z}_{\max}}(\mathbf{s}', \mathbf{s})] + \mathbb{E}_{\mathbf{z}_{\min} \sim \mathcal{U}^d(-1,0)} \mathbb{E}_{p_{\mathbf{z}_{\min}}(\mathbf{s}', \mathbf{s})} [\log p_{\mathbf{z}_{\min}}(\mathbf{s}', \mathbf{s})].$$

[4] The implementation provided by CIC (`https://github.com/rll-research/cic`) is based on the arxiv version (`https://arxiv.org/abs/2202.00161v2`) which is an updated version of the ICLR version (`https://openreview.net/forum?id=kOtkgUGAVTX`). In particular, in the arxiv version, the intrinsic reward only relies on the KNN estimates and the NCE term is only used to train $\phi_s, \phi_z, \psi$. While, the ICLR version includes the NCE term in the intrinsic reward.

And the intrinsic reward is a function of the state-transition and the binary random variable $M$

$$r^{\text{int}}(\mathbf{s}^{(i)}, \mathbf{s}^{(i)'}, M) = \begin{cases} \log\left(c + \frac{1}{k}\sum_k R_k^{(i)}\right) & M = 0, \\ -\log\left(c + \frac{1}{k}\sum_k R_k^{(i)}\right) & M = 1. \end{cases} \tag{9}$$

As in CIC [31], we approximate the joint entropy using Eq. 6.

MOSS's implementation is straightforward; i.e., at each time step, instead of sampling $\mathbf{Z} \sim P(\mathbf{Z})$, we sample $\mathbf{Z} \sim P(\mathbf{Z} \mid M)$ and replace the intrinsic reward with Eq. 9. Pseudocode for MOSS is provided in Alg. 1. In fact, fixing $M = 0$ yields CIC [31].

## 5 Experiments

**Environments** We present the main results of method by evaluating on the Unsupervised Reinforcement Learning Benchmark (URLB) [32], which is a standard unsupervised RL benchmark for continuous control. URLB [32] has three different domains listed in order of increasing difficulty:

- Walker is a biped robot domain constrained to a 2D vertical plane. The challenge for this domain is to learn how to balance and maneuver simultaneously.
- Quadruped is a four-legged robot in 3D space. This domain has a higher dimensional state and action space than the Walker domain.
- Jaco is a 6-DOF robotic arm domain with a three-finger gripper in 3D space that has to learn action constraints and manipulation.

Furthermore, each of the three domains has four different downstream tasks with varying difficulties. We refer readers to Tab 1 for the complete list of 12 tasks. Finally, in OpenAI gym environments [9], since an episode terminates as soon as a robot falls, the agent will naturally avoid falling to the ground as it will get a 0 reward. Hence, during pretraining, the OpenAI gym environment [9] may leak some task information (e.g., standing) to the unsupervised learning agent [31]. Therefore, URLB [32] builds on top of the Deepmind Control Suite [52] to avoid leaking some task information.

The domains in URLB [32] use low-dimensional state information, deterministic transitions, and non-evolving environments. In contrast, ViZDoom [30] uses raw pixel observations, stochastic transition, and evolving environments. Furthermore, since the environment randomly spawns enemies that shoot fireballs, the agent faces external perturbations that surprise the agent. We evaluate our method on the *DefendTheLine* and *TakeCover* maps. Please refer to Figure 2 for environment renders.

(a) URLB based on DMC

(b) The ViZDoom Environment.

Figure 2: **Evaluation Environments.** We evaluate our method in (a) URLB, a standard unsupervised reinforcement learning benchmark based on continuous control (b) ViZDoom a pixel observation based reinforcement learning environment based on the Doom game with multiple maps.

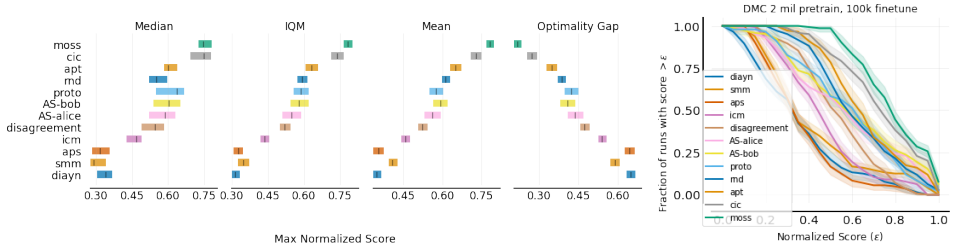
**Evaluation** We follow the benchmark's standard training procedure by pretraining the agent for 2 million steps in each of the three domains and then finetuning the pre-trained agent for $100k$ steps with downstream task rewards. Since AS trains two network-independent policies, we ensure both policies each perform 2 million environment steps during pretraining. To ensure fairness, we pretrain

and finetune each method with the same 12 seeds (0-11) using the code provided in URLB[5] [32]. We present the results in Fig. 3 for a total of 1584 (= 11 algorithms $\times$12 tasks $\times$12 seeds) runs. Our main evaluation metrics are interquartile mean (IQM) and Optimality Gap (OG) of normalized scores, where we used the score of a DDPG [56] agent trained from scratch for 2 million steps as the expert scores [32]. On the one hand, IQM is more robust to outliers than the mean, and IQM is significantly less biased than the median [2]. On the other hand, the Optimality Gap measures how far scores are from the expert scores [2].

**Baselines** Baselines include methods presented in the URLB benchmark [32] as well as CIC [31]. The baseline methods fall into knowledge-based methods, i.e., ICM [42], Disagreement [43], RND [11], data-based methods, i.e., APT [36], Proto-RL [57], AS [19], and competence-based methods, i.e., SMM [33], DIAYN [17], APS [35], CIC [31]. We refer readers to the Appendix of URLB [32] for details on the different baselines. Lastly, since the two policies (Alice and Bob) in AS are parameter-independent, we pretrain for 4 million steps, each policy trained using 2 million steps.

We evaluate our agent in ViZDoom [30] against CIC [31], and a modified version of CIC [31], dubbed NegativeCIC, where the intrinsic reward is the negative joint entropy. Since the ViZDoom environment has high natural environment dynamics entropy, NegativeCIC should act as a strong baseline. We adopt the same pretraining and finetuning procedure as in URBL for ViZDoom.

**MOSS** We keep all hyperparameters as those in CIC [31]. Specifically, unless specified otherwise, we set $M = 0$ for the first half of steps and $M = 1$ for the other half of steps in the episode. We refer readers to the Appendix for more details.




(a) **Aggregate Statistics**. MOSS obtains the highest IQM and the lowest Optimality Gap.

(b) **Performance Profiles.** MOSS stochastically dominates all other methods.

Figure 3: **Main results.** We report aggregate statistics (a) and performance profiles (b) following [2] for 12 downstream tasks on URLB with 12 seeds, providing a total of 144 seeds.

## 5.1 Results

Table 1: **Numerical results.** Following URLB [32] we show the mean and standard error over 12 seeds (0-11). We denote competence-based methods, knowledge-based, and data-based methods in blue, black, and red, respectively.

| Domain | Walker | | | | Quadruped | | | | Jaco | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Task | Flip | Run | Stand | Walk | Jump | Run | Stand | Walk | Bottom Left | Bottom Right | Top Left | Top Right |
| ICM[42] | 381±10 | 180±15 | 868±30 | 568±38 | 337±18 | 221±14 | 452±15 | 234±18 | 112±7 | 94±5 | 90±6 | 93±11 |
| Disagreement [43] | 313±8 | 166±9 | 658±33 | 453±37 | 512±14 | 395±12 | 686±30 | 358±25 | 120±7 | 132±5 | 111±10 | 113±10 |
| RND [11] | 412±18 | 267±18 | 842±19 | 694±26 | **681±11** | 455±7 | 875±25 | 581±42 | 106±6 | 111±6 | 83±7 | 107±5 |
| ICM APT[36] | 596±24 | 491±18 | 949±3 | 850±22 | 508±44 | 390±24 | 676±44 | 464±52 | 114±5 | 120±3 | 116±4 | 114±8 |
| Proto [57] | 378±4 | 225±16 | 828±24 | 610±40 | 426±32 | 310±22 | 702±59 | 348±55 | 130±12 | 131±11 | 134±12 | 146±10 |
| AS-Bob | 475±16 | 247±23 | 917±36 | 675±21 | 449±24 | 285±23 | 594±37 | 353±39 | 116±21 | **166±12** | 143±12 | 139±13 |
| AS-Alice | 491±20 | 211±9 | 868±47 | 655±36 | 415±20 | 296±18 | 590±41 | 337±17 | 109±20 | 141±19 | 140±17 | 129±15 |
| SMM[33] | 428±8 | 345±31 | 924±9 | 731±43 | 271±35 | 222±23 | 388±51 | 167±20 | 52±5 | 55±2 | 53±2 | 57±5 |
| DIAYN [17] | 306±12 | 146±7 | 631±46 | 394±22 | 491±38 | 325±21 | 662±38 | 273±19 | 35±5 | 35±6 | 23±3 | 31±5 |
| APS [35] | 355±18 | 166±15 | 667±56 | 500±40 | 283±22 | 206±16 | 379±31 | 192±17 | 61±6 | 79±12 | 51±5 | 56±7 |
| CIC [31] | 715±40 | **535±25** | **968±2** | 914±12 | 541±31 | 376±19 | 717±46 | 460±36 | 147±8 | 150±6 | 145±9 | 139±9 |
| MOSS (Ours) | **729±40** | 531±20 | 962±3 | **942±5** | 674±11 | **485±6** | **911±11** | **635±36** | **151±5** | 150±5 | **150±5** | **150±6** |

[5]our experiments are based on the code of URLB `https://github.com/rll-research/url_benchmark`, and CIC `https://github.com/rll-research/cic`

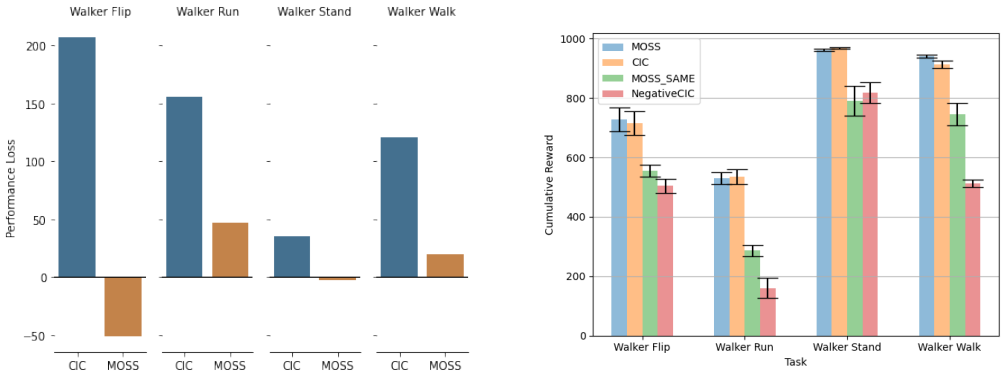
To ensure the reliability of our main results, we report results as described in [2] to account for the uncertainty. We present results in Fig. 3, where, surprisingly, our method obtains state-of-the-art performance on the URLB benchmark [32] on our main evaluation metrics [2]. In particular, the performance profile in Fig. 3b reveals that MOSS stochastically dominates all other methods. We also present numerical results in Tab. 1 for a detailed breakdown over individual tasks.

For the IQM metric, we report an increase of $6\%$ compared to CIC, which is the second leading method. Furthermore, as for the Optimality Gap metric, we decreased $20\%$ compared to CIC. Furthermore, in Fig. 3.b previous methods intersect at multiple points (making it hard to conclude whether a method is better than the other), while MOSS is always above all previous methods. In addition, in Fig. 3.b, MOSS maintains all runs greater than a normalized score as high as $0.5$.

Table 2: **ViZDoom Results**. We report on two maps from ViZDoom to illustrate the robustness of our method in different environments.

| | Defend the Line | Take Cover |
|---|---|---|
| NegativeCIC | 462±11 | 51793±2936 |
| MOSS | **510±22** | **63497±2384** |

We also evaluated our method on ViZDoom [30]. Since the *TakeCover* and *DefendTheLine* maps in ViZDoom provides natural perturbations, we use NegativeCIC as the baseline. We present the numerical results in Tab. 2. Since the environments in ViZDoom are highly stochastic, we remove the top and bottom 10% of scores and report the mean and standard error to account for any outliers. Like in the main results presented earlier, MOSS consistently outperforms the baseline method under this completely new setting.




(a) **Ablation on environment.** Adding stochasticity in the URLB environment hurts CIC. We notice that MOSS is significantly more robust to the stochasticity than CIC.

(b) **Ablation on policy.** Removing the mixture significantly hurts the performance.

Figure 4: **MOSS ablation experiments** on the Walker domain [32]. In (a), we inject stochasticity by adding a Gaussian noise on the agent's action according to a Bernoulli with $p = 0.3$ and report performance loss. In (b), we report performance for CIC, MOSS_SAME, and NegativeCIC. Specifically, MOSS_SAME represents MOSS without the mixture of policies, and NegativeCIC represents CIC with the intrinsic reward as the negative joint entropy.

### 5.2 Ablation

Since MOSS does not make assumptions about the entropy of the environment's dynamics, we are interested in knowing whether adding stochasticity affects performance. For this stochastic setup, during each interaction with the environment, we sample $Y \sim \text{Bernoulli}(0.3)$ and if $Y = 1$, we add a Gaussian Noise $\mathcal{N}(0, 0.2)$ to the agent's intended action. In Fig. 4a, we observe that CIC's [31] performance drops while MOSS' performance is almost unaffected. This finding empirically supports previous works which claim that solely maximizing surprise for unsupervised RL agents is sometimes not enough, especially in stochastic environments [6, 45].

In MOSS, we maintain two different skill distributions, i.e., $\mathbf{Z}_{\max} \sim \mathcal{U}^d(0,1)$ and $\mathbf{Z}_{\min} \sim \mathcal{U}^d(-1,0)$. In particular, setting $\mathbf{Z}_{\min}$ to the same distribution as $\mathbf{Z}_{\max} \sim \mathcal{U}^d(0,1)$ is equivalent to directly optimizing CIC [31] with the two objectives simultaneously. We dub MOSS with the same skill distribution, MOSS_SAME. In Fig. 4b, we observe that naively doing so can result in poor performance, showing the effectiveness of our method in mitigating the simultaneous optimization of two contradicting objectives.

Moreover, as shown in Fig. 4b, NegativeCIC performs poorly on the URLB benchmark [32] since most tasks required structured exploration in a low entropy environment.

## 6 Discussions & Limitations

We presented MOSS, an unsupervised RL method that does not make assumptions about the environment dynamics' entropy by simultaneously maximizing and minimizing surprises. Our method falls into competence-based methods. Leveraging a mixture of policies for more than surprise maximization and minimization is an interesting future research.

**Limitations** One limitation is that our method has a simple and effective deterministic rule for switching between the two objectives *independent* of state or history. We did explore a more sophisticated method in Appendix B that had a better performance but required tuning and extra computation. Therefore, a promising direction is to explore methods that take advantage of state information (e.g., using a meta-controller [29]) to output a more optimal switching mechanism that incentivizes deep structured exploration [44]. Moreover, we only considered competence-based methods. Extending our method to other unsupervised RL methods would be interesting to investigate in future work. Additionally, when finetuning, we transferred network weights which could lead to unlearning of already learned behaviors [12]. It would be meaningful to combine with works that circumvent this issue like [12] which utilized frozen pretrained policies to improve downstream finetuning further. Lastly, we only considered two objectives, and including more meaningful objectives like empowerment or anxiety [39, 50] during pretraining could be an exciting direction.

**Potential Negative Impact** Since we operate under the unsupervised reinforcement learning domain, the agent receives rewards that incentives exploration. Therefore, agents under this domain can be more unpredictable than agents that receive task rewards for intended behaviors. Furthermore, there is no current safeguard for the agent to not act maliciously in its environment; the agent may potentially cause harm to properties situated in the environment, including the agent itself. Therefore, unsupervised learning with safety constraints would be an exciting area of future research to mitigate this potential negative impact.

**Acknowledgement** This work is supported in part by the State Key Laboratory of Intelligent Control and Decision of Complex Systems.

## References

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

[2] Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C Courville, and Marc Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *Advances in Neural Information Processing Systems*, 34, 2021.

[3] Arthur Aubret, Laetitia Matignon, and Salima Hassas. A survey on intrinsic motivation in reinforcement learning. *arXiv preprint arXiv:1908.06976*, 2019.

[4] Kate Baumli, David Warde-Farley, Steven Hansen, and Volodymyr Mnih. Relative variational intrinsic control. *arXiv preprint arXiv:2012.07827*, 2020.

[5] Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. *Advances in neural information processing systems*, 29, 2016.

[6] Glen Berseth, Daniel Geng, Coline Devin, Nicholas Rhinehart, Chelsea Finn, Dinesh Jayaraman, and Sergey Levine. Smirl: Surprise minimizing reinforcement learning in unstable environments. *arXiv preprint arXiv:1912.05510*, 2019.

[7] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

[8] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[9] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[10] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[11] Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*, 2018.

[12] Víctor Campos, Pablo Sprechmann, Steven Hansen, André Barreto, Steven Kapturowski, Alex Vitvitskyi, Adrià Puigdomènech Badia, and Charles Blundell. Coverage as a principle for discovering transferable behavior in reinforcement learning. *CoRR*, abs/2102.13515, 2021.

[13] Víctor Campos, Alexander Trott, Caiming Xiong, Richard Socher, Xavier Giró-i Nieto, and Jordi Torres. Explore, discover and learn: Unsupervised discovery of state-covering skills. In *International Conference on Machine Learning*, pages 1317–1327. PMLR, 2020.

[14] Albin Cassirer, Gabriel Barth-Maron, Eugene Brevdo, Sabela Ramos, Toby Boyd, Thibault Sottiaux, and Manuel Kroiss. Reverb: A framework for experience replay, 2021.

[15] Nuttapong Chentanez, Andrew Barto, and Satinder Singh. Intrinsically motivated reinforcement learning. *Advances in neural information processing systems*, 17, 2004.

[16] Thomas M Cover. *Elements of information theory*. John Wiley & Sons, 1999.

[17] 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.

[18] Benjamin Eysenbach, Ruslan Salakhutdinov, and Sergey Levine. The information geometry of unsupervised reinforcement learning. *arXiv preprint arXiv:2110.02719*, 2021.

[19] Arnaud Fickinger, Natasha Jaques, Samyak Parajuli, Michael Chang, Nicholas Rhinehart, Glen Berseth, Stuart Russell, and Sergey Levine. Explore and control with adversarial surprise. *arXiv preprint arXiv:2107.07394*, 2021.

[20] Karl Friston. The free-energy principle: a unified brain theory? *Nature reviews neuroscience*, 11(2):127–138, 2010.

[21] Karl Friston, Thomas FitzGerald, Francesco Rigoli, Philipp Schwartenbeck, Giovanni Pezzulo, et al. Active inference and learning. *Neuroscience & Biobehavioral Reviews*, 68:862–879, 2016.

[22] Karl Friston, James Kilner, and Lee Harrison. A free energy principle for the brain. *Journal of physiology-Paris*, 100(1-3):70–87, 2006.

[23] Karol Gregor, Danilo Jimenez Rezende, and Daan Wierstra. Variational intrinsic control. *ArXiv*, abs/1611.07507, 2017.

[24] Abhishek Gupta, Benjamin Eysenbach, Chelsea Finn, and Sergey Levine. Unsupervised meta-learning for reinforcement learning. *arXiv preprint arXiv:1806.04640*, 2018.

[25] Michael Gutmann and Aapo Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 297–304. JMLR Workshop and Conference Proceedings, 2010.

[26] Steven Hansen, Will Dabney, Andre Barreto, Tom Van de Wiele, David Warde-Farley, and Volodymyr Mnih. Fast task inference with variational intrinsic successor features. *arXiv preprint arXiv:1906.05030*, 2019.

[27] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. *arXiv preprint arXiv:2111.06377*, 2021.

[28] Tom Hennigan, Trevor Cai, Tamara Norman, and Igor Babuschkin. Haiku: Sonnet for JAX, 2020.

[29] Maximilian Igl, Andrew Gambardella, Jinke He, Nantas Nardelli, N Siddharth, Wendelin Boehmer, and Shimon Whiteson. Multitask soft option learning. In Jonas Peters and David Sontag, editors, *Proceedings of the 36th Conference on Uncertainty in Artificial Intelligence (UAI)*, volume 124 of *Proceedings of Machine Learning Research*, pages 969–978. PMLR, 03–06 Aug 2020.

[30] Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski. Vizdoom: A doom-based ai research platform for visual reinforcement learning. In *2016 IEEE conference on computational intelligence and games (CIG)*, pages 1–8. IEEE, 2016.

[31] Michael Laskin, Hao Liu, Xue Bin Peng, Denis Yarats, Aravind Rajeswaran, and Pieter Abbeel. CIC: Contrastive intrinsic control for unsupervised skill discovery, 2022.

[32] Michael Laskin, Denis Yarats, Hao Liu, Kimin Lee, Albert Zhan, Kevin Lu, Catherine Cang, Lerrel Pinto, and Pieter Abbeel. URLB: Unsupervised reinforcement learning benchmark. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.

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

[34] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[35] Hao Liu and Pieter Abbeel. Aps: Active pretraining with successor features. In *International Conference on Machine Learning*, pages 6736–6747. PMLR, 2021.

[36] Hao Liu and Pieter Abbeel. Behavior from the void: Unsupervised active pre-training. *Advances in Neural Information Processing Systems*, 34, 2021.

[37] Damiano Lombardi and Sanjay Pant. Nonparametric k-nearest-neighbor entropy estimator. *Physical Review E*, 93(1):013310, 2016.

[38] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*, 2018.

[39] Shakir Mohamed and Danilo Jimenez Rezende. Variational information maximisation for intrinsically motivated reinforcement learning. *Advances in neural information processing systems*, 28, 2015.

[40] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[41] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[42] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*, pages 2778–2787. PMLR, 2017.

[43] Deepak Pathak, Dhiraj Gandhi, and Abhinav Gupta. Self-supervised exploration via disagreement. In *International conference on machine learning*, pages 5062–5071. PMLR, 2019.

[44] Miruna Pîslar, David Szepesvari, Georg Ostrovski, Diana Borsa, and Tom Schaul. When should agents explore? *arXiv preprint arXiv:2108.11811*, 2021.

[45] Nicholas Rhinehart, Jenny Wang, Glen Berseth, John Co-Reyes, Danijar Hafner, Chelsea Finn, and Sergey Levine. Information is power: Intrinsic control via information capture. *Advances in Neural Information Processing Systems*, 34, 2021.

[46] Archit Sharma, Michael Ahn, Sergey Levine, Vikash Kumar, Karol Hausman, and Shixiang Gu. Emergent real-world robotic skills via unsupervised off-policy reinforcement learning. *arXiv preprint arXiv:2004.12974*, 2020.

[47] Archit Sharma, Shixiang Gu, Sergey Levine, Vikash Kumar, and Karol Hausman. Dynamics-aware unsupervised discovery of skills. *arXiv preprint arXiv:1907.01657*, 2019.

[48] Harshinder Singh, Neeraj Misra, Vladimir Hnizdo, Adam Fedorowicz, and Eugene Demchuk. Nearest neighbor estimates of entropy. *American journal of mathematical and management sciences*, 23(3-4):301–321, 2003.

[49] DJ Strouse, Kate Baumli, David Warde-Farley, Vlad Mnih, and Steven Hansen. Learning more skills through optimistic exploration. *arXiv preprint arXiv:2107.14226*, 2021.

[50] Chenyu Sun, Hangwei Qian, and Chunyan Miao. From psychological curiosity to artificial curiosity: Curiosity-driven learning in artificial intelligence tasks. *arXiv preprint arXiv:2201.08300*, 2022.

[51] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. 2018.

[52] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

[53] Manan Tomar, Utkarsh A Mishra, Amy Zhang, and Matthew E Taylor. Learning representations for pixel-based control: What matters and why? *arXiv preprint arXiv:2111.07775*, 2021.

[54] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

[55] Kelvin Xu, Siddharth Verma, Chelsea Finn, and Sergey Levine. Continual learning of control primitives: Skill discovery via reset-games. *Advances in Neural Information Processing Systems*, 33:4999–5010, 2020.

[56] Denis Yarats, Rob Fergus, Alessandro Lazaric, and Lerrel Pinto. Mastering visual continuous control: Improved data-augmented reinforcement learning. *arXiv preprint arXiv:2107.09645*, 2021.

[57] Denis Yarats, Rob Fergus, Alessandro Lazaric, and Lerrel Pinto. Reinforcement learning with prototypical representations. In *International Conference on Machine Learning*, pages 11920–11931. PMLR, 2021.

## Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section **??**.
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [Yes]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...

(a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See supplementary material and Section 5. The code is in the supplementary material.
(b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See supplementary material and Section 5
(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] See Section 5
(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See supplementary material.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes]
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] Yes code in the supplementary material
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

# A Additional Implementation Details

## A.1 MOSS Implementation

We implement MOSS using JAX [8], Haiku [28]. We chose to build on top of JAX [8] as we observed a 2x speedup compared to PyTorch [41]. Moreover, we use PyTorch [41] and Reverb [14] to implement the replay buffer. Tab. 3 details the hyper-parameters used in MOSS which are taken directly from CIC [31]. Since MOSS builds on CIC [31], we empirically verified that our implementation on top of JAX [8] matches the performance on top of PyTorch [41].

Training follows the URLB benchmark [32], where an agent is pretrained for 2 million steps and then finetuned on a downstream task for $100k$ steps.

## A.2 Baseline Implementation

For the baselines that were presented in URLB, we obtained the results by running the code from the URLB GitHub repo: `https://github.com/rll-research/url_benchmark`. All hyperparameters were kept the same as the original implementation.

## A.3 Environment

For continuous control domains, we used the custom Deep Mind Control Suite [52] environments from the URL Benchmark GitHub repo: `https://github.com/rll-research/url_benchmark`. For the VizDoom domain, we used code from their official GitHub repo: `https://github.com/mwydmuch/ViZDoom`. We include the environment renders in Figure 2.

## A.4 Double DQN

We made modifications to MOSS to evaluate in discrete action settings. Specifically, we adopted Double-DQN [54] as the backbone reinforcement learning algorithm, which has the Q-value target as

$$Q_t^{\text{target}} = R_{t+1} + \gamma Q(S_{t+1}, _a Q(S_{t+1}, a; \theta), \theta_t^-),$$

Table 3: **Hyperparameters for MOSS and DDPG.** These hyperparameters are fixed throughout all domains.

| DDPG hyper-parameter | Value |
|---|---|
| Replay buffer capacity | $10^6$ |
| Action repeat | 1 |
| Seed frames | 4000 |
| n-step returns | 3 |
| Mini-batch size | 1048 |
| Discount ($\gamma$) | 0.99 |
| Optimizer | Adam |
| Learning rate | $10^{-4}$ |
| Agent update frequency | 2 |
| Critic target EMA rate ($\tau_Q$) | 0.01 |
| Features dim. | 1024 |
| Hidden dim. | 1024 |
| Exploration stddev clip | 0.3 |
| Exploration stddev value | 0.2 |
| Number pre-training frames | $2 \times 10^6$ |
| Number fine-turning frames | $1 \times 10^5$ |
| **MOSS hyper-parameter** | **Value** |
| Skill dim | 64 |
| $\mathbf{Z}_{\max}$ Prior | Uniform[0,1] |
| $\mathbf{Z}_{\min}$ Prior | Uniform[-1,0] |
| Update skill frequency | 50 |
| State net $f_\psi$ | $\dim \mathcal{O} \to 1024 \to 1024 \to 64$ ReLU MLP |
| Skill net $g_{\phi_z}$ | $64 \to 1024 \to 1024 \to 64$ ReLU MLP |
| Prediction net $g_{\phi_s}$ | $64 \to 1024 \to 1024 \to 64$ ReLU MLP |
| Episode partition | 2 |

where $\theta_t$, $\theta_t^-$ are online network parameters and the target network parameters at time $t$, respectively.

Furthermore, since ViZDoom [30] has a smaller action space, we provided the agent with a discrete number of skill embeddings, similar to DIAYN [17]. Moreover, since performing CPC loss [31] in high dimensional pixel space is not ideal, and to save the number of parameters, we use the CNN backbone of the agent's DQN network to project observations into state vectors, then use an MLP identical to the continuous action setting to calculate the CPC loss [31] using the discrete set of skill embeddings. Tab. 4 details the hyper-parameters used for Double DQN and MOSS in the ViZDoom environment.

### A.5 Compute Resources

All experiments were run on an internal cluster with 8 NVIDIA A100 GPU and AMD EPYC 7742 64-Core Processor. Pretraining MOSS takes roughly 5 hours. While finetuning takes roughly 30 mins.

### A.6 Thing that did not work: Skill Distribution

In MOSS, we maintain two different skill distributions, i.e., $\mathbf{Z}_{\max} \sim P(\mathbf{Z} \mid M = 0)$ and $\mathbf{Z}_{\min} \sim P(\mathbf{Z} \mid M = 1)$. In particular, we defined them as $\mathbf{Z}_{\max} \sim \mathcal{U}^d(0, 1)$ and $\mathbf{Z}_{\min} \sim \mathcal{U}^d(-1, 0)$. Given a skill of dimension $d$, another way to define $P(\mathbf{Z} \mid M = 0)$ and $P(\mathbf{Z} \mid M = 1)$ is to allocate half of a skill vector for $m = 0$ and the other half for $m = 1$. In other words, $\mathbf{Z}_{\max}$ and $\mathbf{Z}_{\min}$ are both drawn from $\mathcal{U}^d(0, 1)$, however, when $m = 0$, $(\mathbf{Z}_{\max})_{:d/2} \sim P(\mathbf{Z} \mid M = 0)$ while $(\mathbf{Z}_{\max})_{d/2:} = \mathbf{0}$. Conversely, when $m = 1$, $(\mathbf{Z}_{\min})_{d/2:} \sim P(\mathbf{Z} \mid M = 0)$ while $(\mathbf{Z}_{\max})_{:d/2} = \mathbf{0}$.

Table 4: **Hyperparameters for MOSS and DQN.** These hyperparameters are fixed throughout all domains.

| Double-DQN hyper-parameter | Value |
|---|---|
| Replay buffer capacity | $10^6$ |
| Action repeat | 1 |
| Frame repeat | 12 |
| Seed frames | 4000 |
| n-step returns | 3 |
| Mini-batch size | 1048 |
| Discount ($\gamma$) | 0.99 |
| Optimizer | Adam |
| Learning rate | 0.0001 |
| Agent update frequency | 2 |
| Critic target EMA rate ($\tau_Q$) | 0.01 |
| Hidden dim. | 256 |
| Epsilon Schedule ($\epsilon$) | pretrain: *linear_decay*(start=1.0, end=0.1, steps=$10^5$)<br>finetune: *linear_decay*(start=1.0, end=0.1, steps=$10^4$) |
| Number pre-training frames | $10^6$ |
| Number fine-turning frames | $10^5$ |
| **MOSS hyper-parameter** | **Value** |
| Skill dim | 64 |
| Num skills | 80 |
| Prior | Discrete-Uniform |
| Update skill frequency | 50 |
| Episode partition | 5 |

Table 5: **Adaptive Mode Switching Results on Quadruped**. We report the results of adaptive mode switching MOSS on the quadruped domain with $\beta = 1.1$ obtained from grid search

| Task | MOSS | MOSS$^{\text{adaptive}}$ |
|---|---|---|
| Quadruped Jump | 674±11 | **687±36** |
| Quadruped Run | 485±6 | **512±20** |
| Quadruped Stand | **911±11** | 869±26 |
| Quadruped Walk | 635±36 | **758±37** |

# B Objective Switching

Since our framework has two objectives, the reinforcement learning agent requires collecting experience to train its conditional policy under both proposed objectives. Moreover, the temporal structure of staying in different modes of behavior in humans and animals is not monolithic; therefore, designing when to switch modes for the agent might be an interesting area to consider. For example, previous works investigated a similar setting and used a threshold based on Q-values [44]. However, in unsupervised reinforcement learning, without any sense of the possible downstream task distribution, we wish to design a mode switching mechanism that is both adaptive and scale-invariant across environments.

Literature in active learning proposes a strategy to query high uncertainty samples. We adopt this framework to MOSS. Intuitively, we wish to encourage the agent to collect more trajectories at places with higher *epistemic uncertainty* for the agent to learn more in a more unfamiliar situation. Since we do not wish to train additional networks, we use the online critic network to output a sample variance to act as a proxy for uncertainty. Concretely, for a given mode $m$, we sample a batch $N$ of skill-vectors $z_m^{(i)} \sim p(z|m)$. We then calculate the sample variance as

$$\mathbb{V}_{t,m}(Q_m^\pi) = \frac{\sum_{i=1}^{N} (Q(s_t, \pi(\cdot|s_t, z_m^{(i)}), z_m^{(i)}) - \bar{Q}_m^\pi)^2}{N-1},$$

where $\bar{Q}_m^\pi$ is the sample mean Q-value for the batch.

However, since variance is a measurement with units, and we do not wish to introduce additional environment-specific hyperparameters, we use the history of variance and the current Q-value variance as function inputs to select modes, bypassing the unit sensitivity problem across environments. Concretely, we keep a record of the sample variance of both modes at each time step $t$, denoted as $\mathbb{V}_{t,M=0}$ for maximization skills and $\mathbb{V}_{t,M=1}$ for minimization skills. At each skill switching interval, the agent selects the mode $m_t$ for the current time step by,

$$m_t = \begin{cases} -(m_{t-k}-1) & \beta * \mathbb{V}_{t-k,-(m_{t-k}-1)} \leq \mathbb{V}_{t,-(m_{t-k}-1)} \\ m_{t-k} & \text{otherwise,} \end{cases}$$

where $\beta$ is a coefficient greater than 1 that controls the threshold of switching modes, and $k$ is the interval we use to switch skills and modes. Intuitively, we wish to switch to the opposite mode if the opposite mode's uncertainty at time $t$ is greater than a coefficient times the last recorded uncertainty. We show some preliminary results for the quadruped domain in Tab. 5. The effectiveness of adaptive methods is demonstrated in the results, where MOSS$^{\text{adaptive}}$ was able to beat out MOSS. However, this method requires memory and computational resources and we found that the hyperparameter $\beta$ is somewhat sensitive across domains and requires tuning. Therefore, we leave additional investigations for mode switching to future work because presenting an efficient and hyperparameter-insensitive method of maximization and minimization of surprise was the main scope of this paper.

# C Additional Results

## C.1 Prior Distribution $P(M)$

In MOSS, we deterministically set $M = 0$ for the first half of the steps and $M = 1$ for the other half of the steps in the episode. This corresponds to having $50\%$ of maximization data and $50\%$ of minimization data. We report additional results on Walker for different ratios of maximization and minimization in Tab. 6.

Table 6: **Ablation on prior** $M$. A prior towards entropy maximization results in a better performance than a prior towards entropy minimization.

| | Walker Flip | Run | Stand | Walk |
|---|---|---|---|---|
| 30% maximization | 723±36 | 515±32 | 967±3 | 921±18 |
| 40% | 738 ±42 | 526±15 | 968±2 | 908±24 |
| 50% (MOSS) | 729±40 | 531±20 | 962±3 | 942±5 |
| 60% | 781±38 | 599±11 | 967±1 | 935±6 |
| 70% | 785±39 | 567±15 | 965±3 | 933±7 |

We observe that a prior towards entropy maximization results in a better performance than a prior towards entropy minimization on Walker.

## C.2 Skill Prior

We present additional results on the dimensionality of the skill vector along with the event space (discrete vs continuous) in Tab. 7. We find that a continuous skill performs better than a discrete skill but not by a large margin. However, it seems that in general, a higher dimensional skill performs better, especially in Quadruped which has a higher dimensional state and action space than the Walker domain. Our results suggest that the optimal skill vector may be task-dependent.

## C.3 Zero-shot Results

We show zero-shot results of different URL methods in Tab. 8. In addition, Tab. 9 provides zero-shot results for both $\mathbf{Z}_{\max}$ and $\mathbf{Z}_{\min}$.

Table 7: **Ablation on prior** $Z$. A continuous skill performs better than a discrete skill but not by a large margin. In addition, a higher dimensional skill tends to perform better.

| | Continuous 1 | Continuous 16 | Continuous 64 (MOSS) | Discrete 1 |
|---|---|---|---|---|
| Walker Flip | 827±50 | 934±12 | 729±40 | 672±9 |
| Walker Run | 329±41 | 245±44 | 531±20 | 335±20 |
| Walker Stand | 930±7 | 830±25 | 962±3 | 931±8 |
| Walker Walk | 954±4 | 960±2 | 942±5 | 751±70 |
| Quadruped Jump | 193±27 | 649±25 | 674±11 | 194±50 |
| Quadruped Run | 167±17 | 444±26 | 485±6 | 162±20 |
| Quadruped Stand | 306±32 | 834±33 | 911±11 | 267±38 |
| Quadruped Walk | 158±19 | 569±49 | 635±36 | 217±23 |

### C.4 Finetune Learning Curves

We include the finetune learning curves in Figure 5 for MOSS and two methods that are most related to MOSS, namely, CIC and Adversarial Surprise.

We can see that these methods generally improved with finetuning with task reward. However, we observe that the Quadruped domain had a noticeably flatter learning curve, meaning that methods benefited less from training than in other domains.

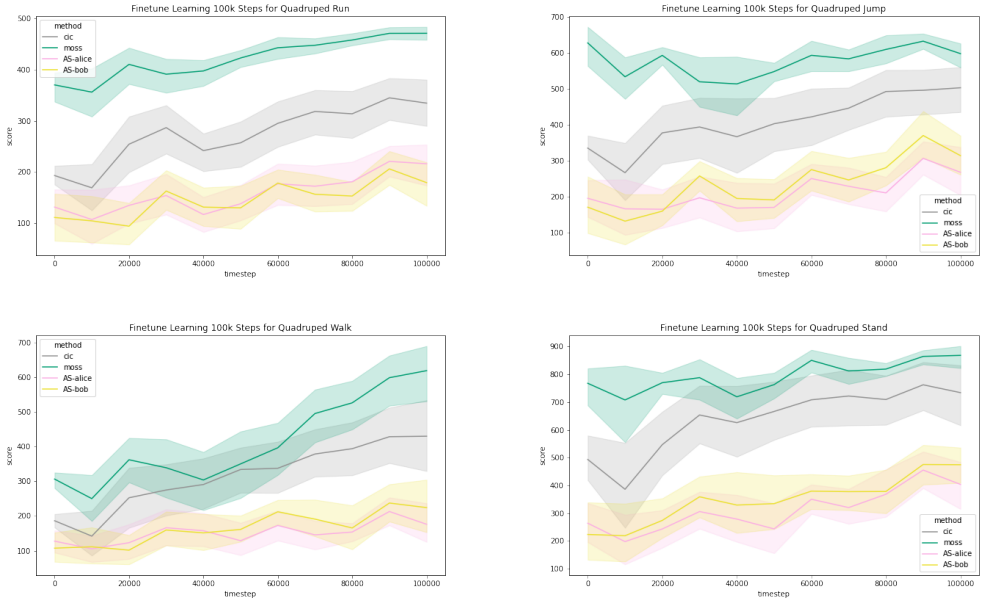


Table 8: **Zero-shot results.** The zero-shot performance of methods on the URL Benchmark

| Domain | Walker | | | | Quadruped | | | | Jaco | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Task | Flip | Run | Stand | Walk | Jump | Run | Stand | Walk | Reach Bottom Left | Reach Bottom Right | Reach Top Left | Reach Top Right |
| ICM | 78±3 | 32±1 | 170±6 | 60±4 | 225±33 | 150±22 | 299±44 | 153±23 | **9±2** | **7±2** | 21±3 | **10±3** |
| Disagreement | 207±2 | 78±0 | 366±3 | 167±2 | 464±10 | 269±6 | 534±11 | 273±7 | 5±1 | 3±1 | **27±4** | 9±2 |
| RND | **237±3** | 89±1 | 392±4 | 195±3 | 549±7 | 319±4 | 638±6 | 321±9 | 4±1 | 3±1 | 17±2 | 4±0 |
| APT | 235±3 | **91±0** | **401±3** | **204±3** | 304±38 | 194±22 | 384±44 | 198±24 | 1±1 | 0±0 | 0±0 | 0±0 |
| Proto | 171±4 | 72±2 | 322±9 | 162±6 | 36±5 | 23±3 | 46±6 | 23±3 | 1±0 | 1±0 | 10±2 | 4±1 |
| AS-BOB | 35±3 | 22±0 | 132±4 | 28±2 | 170±37 | 111±24 | 223±49 | 107±21 | 1±1 | 0±0 | 1±1 | 0±0 |
| AS-ALICE | 31±2 | 24±1 | 139±9 | 28±2 | 195±26 | 131±17 | 264±34 | 127±16 | 0±0 | 0±0 | 0±0 | 0±0 |
| SMM | 117±11 | 49±4 | 229±13 | 100±16 | 73±23 | 46±14 | 91±27 | 48±15 | 1±0 | 4±2 | 9±2 | 6±1 |
| DIAYN | 22±2 | 18±1 | 107±9 | 20±2 | 136±31 | 91±20 | 181±41 | 93±21 | 1±0 | 2±1 | 3±1 | 4±1 |
| APS | 35±2 | 27±1 | 162±11 | 30±2 | 134±20 | 89±13 | 180±26 | 89±13 | 0±0 | 0±0 | 0±0 | 0±0 |
| CIC | 218±5 | 79±1 | 356±6 | 174±4 | 415±34 | 248±21 | 493±41 | 250±23 | 0±0 | 0±0 | 0±0 | 1±0 |
| MOSS | 166±5 | 58±1 | 295±9 | 112±3 | **627±28** | **370±16** | **767±35** | **306±12** | 0±0 | 2±1 | 0±0 | 1±0 |

Table 9: **Zero-shot results of $\mathbf{Z}_{\max}$ and $\mathbf{Z}_{\min}$**. The comparison of zero-shot results of the two mixture of skills evaluated on the Quadruped domain.

| | Quadruped Jump | Quadruped run | Quadruped Stand | Quadruped Walk |
|---|---|---|---|---|
| $\text{MOSS}_{\text{min,frozen}}$ | 85.3±9.5 | 45±4 | 110±13 | 43±4 |
| $\text{MOSS}_{\text{max,frozen}}$ | 627±28 | 370±16 | 767±35 | 306±12 |

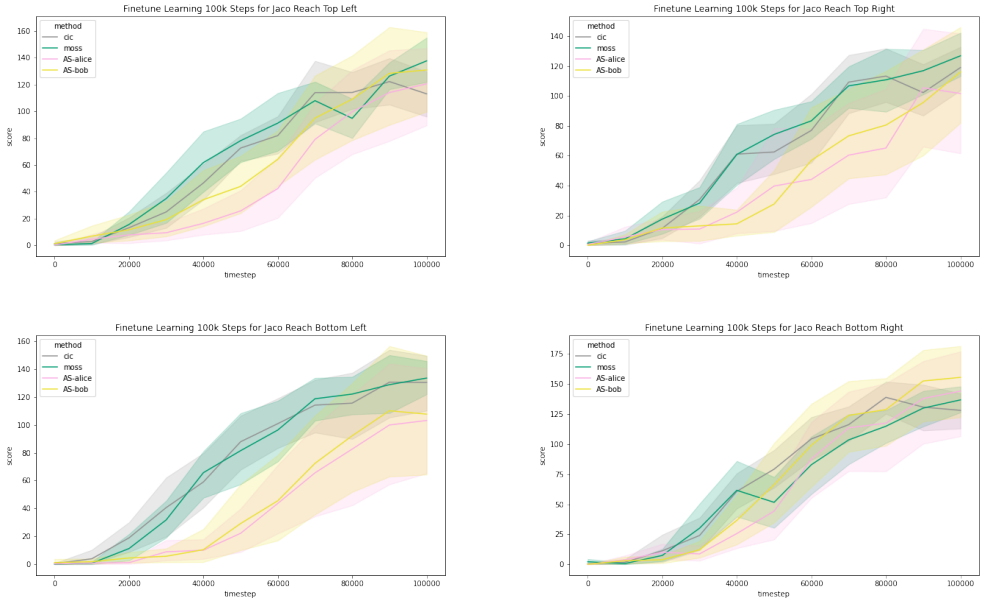



Figure 5: Finetune Learning Curves