# ASPiRe: Adaptive Skill Priors for Reinforcement Learning

**Mengda Xu** [1, 2], **Manuela Veloso** [2,3], **Shuran Song** [1]
[1] Department of Computer Science, Columbia University
[2] J.P. Morgan AI Research [3] School of Computer Science, Carnegie Mellon University (emeritus)
https://aspire.cs.columbia.edu/

## Abstract

We introduce ASPiRe (Adaptive Skill Prior for RL), a new approach that leverages prior experience to accelerate reinforcement learning. Unlike existing methods that learn a single skill prior from a large and diverse dataset, our framework learns a library of different distinction skill priors (i.e., behavior priors) from a collection of specialized datasets, and learns how to combine them to solve a new task. This formulation allows the algorithm to acquire a set of specialized skill priors that are more reusable for downstream tasks; however, it also brings up additional challenges of how to effectively combine these unstructured sets of skill priors to form a new prior for new tasks. Specifically, it requires the agent not only to identify which skill prior(s) to use but also how to combine them (either sequentially or concurrently) to form a new prior. To achieve this goal, ASPiRe includes Adaptive Weight Module (AWM) that learns to infer an adaptive weight assignment between different skill priors and uses them to guide policy learning for downstream tasks via weighted Kullback-Leibler divergences. Our experiments demonstrate that ASPiRe can significantly accelerate the learning of new downstream tasks in the presence of multiple priors and show improvement on competitive baselines.

## 1 Introduction

Transferring prior experience to new tasks is central to an agent's adaptability. In this work, we aim to accelerate online reinforcement learning by leveraging prior experience from large offline data. Previous works [1, 2, 3] have proposed to extract temporally extended actions into *skills* from offline datasets and learn a single behavior prior over the skill space, which we refer to as *skill prior*. The skill prior is then used to guide the learning for a new downstream task by constraining the learned policy to stay close to the prior distribution. However, regularizing with a single skill prior might be insufficient for more complex downstream tasks. We conclude two major limitations for using a single skill prior: (1) The datasets could be generated by multiple policies for diverse tasks. As a consequence, the action distribution from behavior trajectories might be multi-modal, and the resulting learned prior might learn an "average" behavior [4] (2) Complex tasks not only require reusing previous skills but also composing them. Consider a task that requires the agent to traverse a maze with obstacles inside. The agent needs to compose navigation skills and avoid skills concurrently when the obstacle is observed (See Fig. 1) and only activate navigation skills when it is not. Guided with only navigation skill prior or avoid skill prior might lead to inefficient learning.

This observation suggests guiding the learning with multiple skill priors, with each prior specialized in one task. During the learning of new downstream tasks, the most relevant priors should be selected to regularize the policy based on the task and specific states. Specifically, it requires the agent to: (1) **Identify which skill prior(s) to use.** Since not all skill priors are relevant to the new task at hand, imposing irrelevant skill priors to regularize can mislead the learning process. (2) **Learn how**

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

**to combine multiple skill priors.** Many complex tasks require the agent to identify and combine multiple priors to solve them effectively. The integration of skill priors can be either **sequential** (active one prior at a time) or **concurrent** (active multiple priors at the same time), which need to be determined by the agent automatically.

To address the above challenges, we propose **ASPiRe**, a new method to accelerate the learning of new downstream tasks by guiding the policy with an adaptive skill prior. To implement this idea, we first extract the multiple primitive skill priors from the separated and labeled offline data. Each primitive skill prior is specialized in solving one specific task (e.g., navigation prior and avoid prior). For a new downstream task, the policy is trained with a new task reward and regularized by the adaptive skill prior. The adaptive skill prior can be one of the primitive skill priors or constructed by composing primitive skill priors.

Our method includes an Adaptive Weight Module (AWM) that infers an adaptive weight assignment over learned primitive skill priors and uses these inferred weights to guide policy learning through weighted Kullback-Leibler (KL) divergence. The weight applied to the KL divergence between the learned policy and a given primitive skill prior determines its impact on the learned policy. This implicitly composes multiple primitive skill priors into a *composite skill prior* and guides the policy learning. Our formulation allows the algorithm to combine the primitive skill priors both sequentially and concurrently as well as adaptively adjust them during the online learning phase.

We evaluate our method in challenging tasks with sparse rewards. We demonstrate that our method can accelerate the learning of new downstream tasks in the presence of multiple priors and show improvement over competitive baselines on both learning efficiency and task success rate.

## 2 Related work

**Offline reinforcement learning.** Offline reinforcement learning [4] exploits an existing offline dataset and learns the best policy from it without interacting with environments. This is valuable when interacting with the environment for a task is expensive. Many algorithms have been developed, both in model-free [5, 6, 7, 8, 9, 10, 11, 12, 13] and model-based [14, 15, 16, 17, 18] settings.

Offline RL has also demonstrated its ability to accelerate online learning [2, 3, 19]. While most of the previous works only consider a behavior prior for a single task, the policy is guided by the behavior prior to achieve efficient exploration. Rao et al. [20] proposes to use a mixture latent variable model to compose multiple behavior priors, but this framework only activates one behavior prior at a time. How to leverage multiple behavior priors during online learning both sequentially and concurrently remains an open question. Our method provides a solution to adaptively select a behavior prior (i.e., skill prior) or compose a new prior to guide the policy learning.

**Reusing and Composing skills.** It is crucial for intelligent agents to reuse and compose learned skills to solve new tasks. A wide range of works have been proposed to solve this longstanding problem [21, 22]. One way to compose and reuse skills is using value functions [23, 24, 25] or cumulants [26, 27, 28, 29]. Numerous works also explore the hierarchical structure in which a high-level policy integrates a collection of low-level controllers (primitives) [30, 31, 32, 33, 34] and each of low-level controllers contains a distinct skill. However, most of the hierarchical methods only sequentially activate the primitives. The other line of work for reusing and composing skills is to operate on a latent space, which can be mapped to action space [35, 36, 37, 38, 39, 40]. Among those works, several approaches employ the idea of mixture-of-experts [41] by explicitly modeling a library of primitive skills and learning a weight gate function to compose the skills [42, 43, 42, 44]. These works show that this additional structure is beneficial for transferring skills to complex tasks. Our work can also be viewed through the mixture-of-experts lens, i.e., solve downstream tasks by composing a library of primitives. However, our work treats the composition as a regularization to guide the policy learning, instead of executing the composition as a policy directly.

## 3 Approach

The goal for ASPiRe is to accelerate downstream task learning using a library of primitive skill priors. There are two main challenges for this task: 1) Decide which primitive skill prior should be activated given a new downstream task, where using unrelated primitive skill prior could mislead the learning

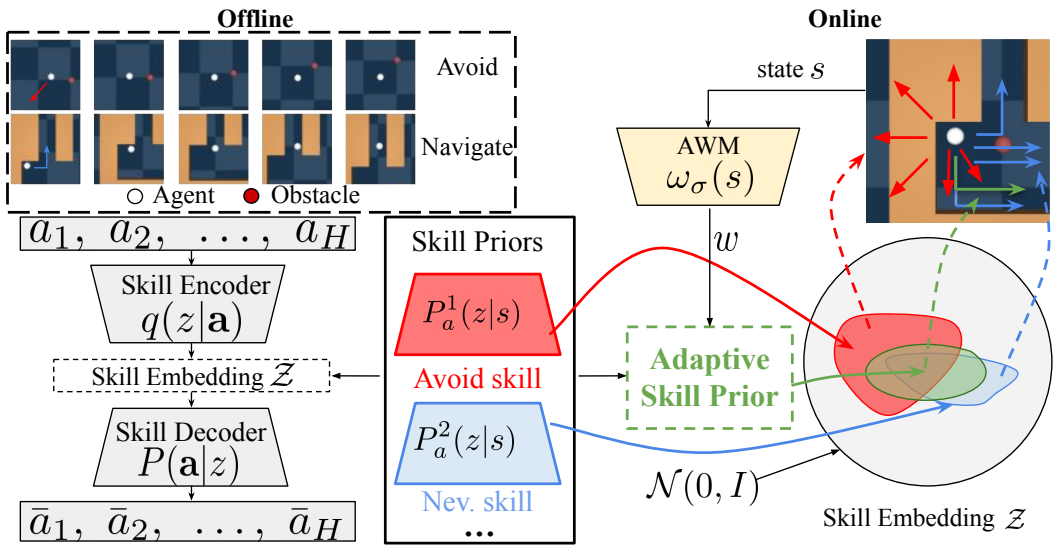


Figure 1: **ASPiRe overview.** In offline, we learn a shared skill embedding space and a collection of distinct primitive skill priors, e.g. , avoid and navigation primitives are learned. The skill encoder embeds an action sequence $\mathbf{a}$, which is sampled from aggregated dataset, into a latent vector in embedding $\mathcal{Z}$ and the action sequence get reconstructed by the skill decoder. The primitive skill prior $P_a^i(z|s)$ imitates the state-skill (in embedding space) pairs in dataset $\mathcal{D}_i$. In online stage, ASPiRe composes multiple primitive skill priors via weighted KL divergence, where weighs are inferred by Adaptive Weight Module (AWM).

process. 2) How to best combine multiple primitive skill priors, either sequentially or concurrently. In particular, to the best of our knowledge, how to simultaneously constrain the policy learning with multiple skill priors is an open challenge.

To address these challenges, we propose a method that regularizes the policy learning using an "adaptive" skill prior, which can be either one of the primitive skill priors or a newly created skill prior by the composition of multiple primitive skill priors (i.e., the composite skill prior).

Our approach consists of two learning stages: offline and online (See Fig. 1). In the offline stage, the system learns multiple primitive skill priors from a set of labeled datasets. Each dataset contains expert demonstrations of one specific primitive task. In the online stage (for downstream tasks), the algorithm creates an adaptive skill prior by assigning weight over primitive skill priors. The weights are generated by the adaptive weight module (AWM). Then they are used to regularize the policy training. In the running example in Fig. 1, the policy is regularized by avoid and navigation primitives **concurrently** when the obstacle is observed.

**Problem Formulation:** In the offline learning stage, the algorithm has access to K datasets $\{\mathcal{D}_1, ..., \mathcal{D}_K\}$ with task labelling $\{\mathcal{T}_0, .., \mathcal{T}_K\}$. Each dataset $\mathcal{D}_i$ contains action-state trajectories $\tau = \{(s_0, a_0), ..., (s_T, a_T)\}$ for task $\mathcal{T}_i$ (i.e. push). The datasets are generated by expert behavior policy $\mu_1, .., \mu_K$. We further assume that none of the single behaviors can individually solve the typically more complex downstream learning tasks.

The online learning stage for the downstream task is formulated as a reinforcement learning problem, modelled as an infinite-horizon Markov Decision Process (MDP) $\mathcal{M} = (\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \rho, \gamma)$, where $\mathcal{S}$ is the continuous state space, $\mathcal{A}$ is the continuous action space, $\mathcal{P}$ is the dynamic $p(s_{t+1}|s_t, a_t)$, $\mathcal{R}$ is the scalar reward $r_t$, $\rho$ is the initial state distribution and $\gamma$ is the discount factor. The goal is to learn a policy $\pi(a_t|s_t)$ to maximize the expected discounted sum of rewards $\sum_{t=0}^{T} \mathbb{E}_\pi[\gamma^t r(s_t, a_t)]$.

### 3.1 Extracting multiple primitive skill Priors

We denote an action sequence $\{a_t, ..., a_{t+H-1}\}$ with fixed horizon $H$ as a skill $\mathbf{a}_t$. In the offline learning phase, we first learn a shared low-dimensional skill embedding space $\mathcal{Z}$ from aggregated

datasets $\{\mathcal{D}_i\}_{i=0}^K$. We then construct primitive skill priors on top of the skill embedding space for each dataset $\mathcal{D}_i$. Distinct primitive skill priors can generate different types of skills based on the task associated with $\mathcal{D}_i$.

To learn the skill embedding, following the previous works [1, 2, 45], we fit a variational autoencoder and maximize the following evidence lower bound (ELBO):

$$\log p(\mathbf{a}_t) \geq \mathbb{E}_q \Big[ \underbrace{\log p(\mathbf{a}_t|z)}_{\text{reconstruction}} - \beta \big( \underbrace{\log q(z|\mathbf{a}_t) - \log p(z)}_{\text{regularization}} \big) \Big]. \tag{1}$$

Here, $\beta$ is a parameter that controls the regularization term. The skill encoder $q(z|\mathbf{a}_t)$ embeds the skill $\mathbf{a}_t$ into latent space $\mathcal{Z}$ and the skill decoder $p(\mathbf{a}_t|z)$ reconstructs the latent vectors back to skills. We set $p(z)$ as a unit Gaussian prior to regularize the encoder learning. Note that action sequences $\mathbf{a}$ are randomly sampled from all offline dataset $\{\mathcal{D}_i\}_{i=0}^K$ such that diverse action sequences can be embedded in a shared skill embedding space with the same encoder and decoder.

We freeze the skill encoder and decoder parameters after sufficient training. Next, we extract multiple primitive skill priors on top of the shared skill embedding. For each offline dataset $\mathcal{D}_i$, we extract a primitive skill prior $P_a^i(z|s_t)$, which is parameterized as a Gaussian distribution. To train primitives, we sample state and skill tuples $(s_t, \mathbf{a}_t)$ from $\mathcal{D}_i$ and minimize the KL divergence between the skill posterior and the predicted prior $\mathbb{E}_{(s_t,\mathbf{a}_t)\sim\mathcal{D}_i} D_{\text{KL}}\big(q(z|\mathbf{a}_t), p_a^i(z|s_t)\big)$, similar to previous work [2].

### 3.2 Regularized reinforcement learning via adaptive skill prior

For downstream task learning, instead of learning a policy on action space $\mathcal{A}$, we learn a policy $\pi_\theta(z|s_t)$ which acts on high-level skill embedding space $\mathcal{Z}$. Previous work [2] has demonstrated that learning on the skill embedding space is beneficial for complex long-horizon tasks. The skill embedding $z$ can be decoded into an action sequence $\mathbf{a}$ via the skill decoder $p(\mathbf{a}|z)$ for $H$ steps action execution. We further replace the dynamic in our problem formulation with H-step dynamic $p'(s_{t+H}|s_t, z_t)$ and rewards with H-step rewards $r'_t = \sum_{t=1}^H r_t$.

We guide the policy learning via an adaptive skill prior over the course of online learning for downstream tasks. Ideally, the policy should be regularized by all relevant primitive skill priors at each state. We consider two scenarios: 1) only one primitive prior is relevant, 2) multiple primitive priors are relevant. Previous work [2] considers the first scenario and regularizes the learning policy by providing an additional negated KL divergence between the policy and the offline skill prior. Similarly, SVPG [46] regularizes the policy parameters by a prior distribution and optimize the object via SVGD [47].

In states where multiple primitive skills are involved, the agent needs to achieve several goals simultaneously, which are implicitly encoded in distinct primitive skill priors. It is therefore natural to guide the policy learning with multiple primitive skill priors. To implement this idea, we can augment the standard RL objective with a negated *weighted KL divergence*:

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_\pi \Big[ r(s_t, z_t) - \alpha \underbrace{\sum_{i=1}^{K} \omega_i(s_t) D_{KL}(\pi(z_t|s_t), p_a^i(z_t|s_t))}_{\text{weighted KL divergence}} \Big] \tag{2}$$

Here, we refer to $\omega(s)$ as the *weighting function*, which specifies the weights $w = (w_1, w_2, ..., w_K)$ to apply on KL divergence between the policy and primitive skill priors. We consider weights are bounded $\{w_i\}_{i=1}^K \in [0, 1]$ and are normalized $\sum_{i=1}^K w_i = 1$. Assigning weights to the primitive skill priors implicitly forms a *composite skill prior*. This can be interpreted as the learned policy will be constrained by a skill distribution $p_c(z_t)$ that can minimize the weighted KL divergence given weights $w$ at state $s_t$:

$$p_c(z_t) = \arg\min_{p_c} \sum_{i=1}^{K} w_i D_{KL}(p_c(z_t), p_a^i(z_t|s_t)) \tag{3}$$

We refer to it as *composite skill prior* generated by weight $w$ at state $s_t$. Weights at a given state can be viewed as a measure of how much one primitive skill prior should contribute to the composite skill prior. The state-dependent weights can be learned simultaneously with the policy, and we discuss it in section 3.3. For now, we assume the weighting function $\omega(s_t)$ is given, and the goal is to maximize the Equation 2.

We follow the policy constraints formulation [5, 6, 10] to optimize the objective. We consider a Q-function $Q_\phi$ and a policy $\pi_\theta$ parameterized by the network with parameters $\phi$ and $\theta$, respectively. The policy parameters can be optimized by minimizing the weighted KL divergence penalty:

$$J_\pi(\theta) = \mathbb{E}_{s_t \sim D}\Big[\mathbb{E}_{z_t \sim \pi_\theta}\big[-Q_\phi(s_t, z_t) + \alpha \sum_{i=1}^{K} \omega_i(s_t) D_{KL}(\pi_\theta(z_t|s_t), p_a^i(z_t|s_t))\big]\Big] \tag{4}$$

Here $\alpha \geq 0$ is a temperature parameter to constrain the weighted KL-divergence, which can be tuned automatically by providing a target divergence parameter $\delta$ [2] (See appendix A.3 for details). Q-function parameters can be optimized by minimizing the Bellman residual:

$$J_Q(\phi) = \mathbb{E}_{s_t, z_t \sim D}\Big[\frac{1}{2}\big(Q_\phi(s_t, z_t) - (r(s_t, z_t) + \gamma Q_{\bar{\phi}}(s_{t+1}, \pi_\theta(z_{t+1}|s_{t+1}))\big)^2\Big] \tag{5}$$

### 3.3 Adaptive Weight Module

In this section, we introduce Adaptive Weight Module (AWM), a component to infer the optimal weights adaptively over the course of tasks. Our goal is to learn a parameterized weighting function $\omega_\sigma$ to compose the primitive skill priors via weighted KL divergence, and the resulting composite skill prior can guide the policy learning. We evaluate the expected critic value of skills under the composite skill prior distribution generated by the weights $\omega$. The weights leading to the highest critic value is selected to assign among primitives. Both the weighting function $\omega_\sigma$ and the skill prior generator $G_\eta$ are learned online along with the policy $\pi_\theta$.

**Learning skill prior generator.** To efficiently evaluate the skills under composite skill priors, we implement skill prior generator $G_\eta(z|s_t, w)$, a generative model which outputs the composite skill prior generated by the weights $w$ given the state $s_t$. This can be optimized by minimizing the weighted KL divergence between output distribution $G_\eta(\cdot|s_t, w)$ and primitive skill priors:

$$J_G(\eta) = \mathbb{E}_{s_t \sim \mathcal{D}} \sum_{i=1}^{K} w_i D_{KL}(G_\eta(z|s_t, w), P_a^i(z_t|s_t)) \tag{6}$$

We parameterize the skill prior generator as a Gaussian distribution by the network with parameters $\eta$. During the learning, we randomly sample weights $w$ from the unit simplex as the weight input. This helps the skill prior generator to learn all possible compositions across states.

**Learning adaptive skill prior.** Our goal is to construct an adaptive skill prior that is biased towards the high reward trajectories, similar to Siegel et al. [10]. The key difference is that our approach aims to construct such adaptive prior for a novel downstream task by composing multiple primitive skill priors. In effect, we would like to find weights assignment such that the corresponding composite skill prior can contain the skills that are more likely to lead to task success. We formulate this as:

$$\sigma = \arg\max_{\sigma} \sum_{t} \mathbb{E}_{s_t \sim \mathcal{D}}\big[\mathbb{E}_{z \sim G_\eta(\cdot|s_t, \omega_\sigma(s_t))}[Q_\phi(s_t, z)]\big] \tag{7}$$

where $Q_\phi$ is the policy critic defined in section 3.2. The Q-value in the objective measures the usefulness of skills for the task at hand. The inner expectation can be viewed as a measurement of the usefulness of the composite skill prior generated by weight $w$. The objective prefers a composite skill prior, in which skills have higher Q-value and are more promising to be explored. We approximate the inner expectation with $M$ samples from $G_\eta$, i.e, $\mathbb{E}_{z \sim G_\eta(\cdot|s_t, \omega_\sigma(s_t))}[Q_\phi(s_t, z)] = \frac{1}{M}[Q_\phi(s_t, z)|z \sim G_\eta(\cdot|s_t, \omega_\sigma(s_t))]$. We find $M = 20$ is sufficient to provide a stable and accurate estimation.

We parameterize the weighting function $\omega_\sigma$ by the network with parameter $\sigma$, and use softmax as the output layer. To learn the optimal weight, we set the Equation 7 as the loss function. We add an

**Algorithm 1** ASPiRe Algorithm

**Inputs:** K primitive skill priors $\{P_a^i(z_t|s_t\}_{i=0}^K$
**Initialize:** replay buffer $\mathcal{D}$, policy $\pi_\theta(z_t|s_t)$, critic $Q_\phi(s_t, z_t)$, target critic $Q_{\bar{\phi}}(s_t, z_t)$, weighting function $\omega_\sigma(s_t)$, skill prior generator $G_\eta(z_t|s_t, w_t)$
**for** each iteration **do**
  **for** every H environment steps **do**
    $z_t \sim \pi_\theta(z_t|s_t)$ ▷ Sample skill from policy
    $s_t' \sim p(s_{t+H}|s_t, z_t)$. ▷ Execute skill
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{s_t, z_t, r_t, s_t'\}$ ▷ Store experience in replay buffer
  **end for**
  **for** each gradient step **do**
    Sample a batch of transition tuples $\{s_t, z_t, r_t, s_t'\}$ from $\mathcal{D}$
    $w_t = \omega_\sigma(s_t)$ ▷ Get weights from weighting function
    $\theta \leftarrow \theta - \lambda_\theta \nabla_\theta J_\pi(\theta)$ ▷ Update policy parameters via weighted KL divergence
    $\phi \leftarrow \phi - \lambda_\phi \nabla_\phi J_Q(\phi)$ ▷ Update critic parameters
    $\eta \leftarrow \eta - \lambda_\eta \nabla_\sigma J(\eta)$ ▷ Update generator parameters
    $\sigma \leftarrow \sigma - \lambda_\sigma \nabla_\sigma J(\sigma)$ ▷ Update weighting parameters
    $\bar{\phi} \leftarrow \tau\phi + (1-\tau)\bar{\phi}$ ▷ Update target critic
  **end for**
**end for**

additional regularization term to penalize weights for deviating from uniform weight $\frac{1}{K}$ in the early stage of the training. This encourages the learned policy to explore more skills in diverse primitive skill priors and prevents weights from collapsing to a singleton due to the noise in Q-value in the early phase. Here, $\beta$ is the annealing coefficient, and it gradually decreases. The weighting function parameters can be trained to minimize the following:

$$J_\omega(\sigma) = \mathbb{E}_{s_t \sim \mathcal{D}}\Big[\mathbb{E}_{z \sim G_\eta(\cdot|s_t, \omega_\sigma(s_t))}\Big[-Q_\phi(s_t, z) + \beta||\omega_\sigma(s_t) - \frac{1}{K}||_2^2\Big]\Big] \tag{8}$$

For each gradient step, we optimize both the policy and the AWM. The full Adaptive Skill Prior algorithm is summarized in Algorithm 1. For implementation details, see appendix, A.4.

# 4 Experiments

We would like to answer the following questions through our experiments: a) Can leveraging multiple primitive skill priors accelerate downstream task learning? b) What's the advantage of using the composite skills as regularization compared to directly using them as policy? c) Can AWM optimally assign proper weights over different primitive skill priors based on the downstream tasks, even when irrelevant primitive skill priors are presented?

## 4.1 Experiment setup

We evaluate our method in three modified environments from D4RL [48]. ASPiRe is given two primitive skill priors in the first two experiments. For the last experiment, we test ASPiRe's ability to compose three primitive skill priors. For each environment, we first collect datasets for distinct skills from expert demonstrations, which allows us to extract distinct primitive skill priors. We then evaluate our method with extracted primitives in downstream tasks.

**Point Maze.** We modified the Point Maze environment by adding multiple obstacles in the maze. The task is to navigate a point mass agent in the maze and reach the goal. During this process, the agent must avoid the obstacles randomly placed in the maze. The original point maze environment is already challenging because of the sparse reward feedback. Randomly placing obstacles in the maze makes this problem even more difficult. For each episode, the maze layout is fixed. However, the agent start, goal, and obstacles locations are randomly specified. Our method will be equipped with two primitive skill priors for this task: navigation and avoid.

**Ant Push.** The ant needs to push the box to the goal while avoiding an obstacle in this task. ASPiRe will carry two primitive skill priors: push and avoid. The ant, box, obstacle, and goal locations are

randomly generated at the beginning of every episode. The agent will receive a sparse reward upon the box reaching the goal location.

**Ant Maze.** We modified the Ant Maze by adding an obstacle and a box in the maze. The agent's goal is to traverse the maze while pushing the box to the goal position without hitting the obstacle. The agent always starts in the lower right corner of the maze. However, the box, goal, and obstacle positions are randomly selected from a pre-defined set. Ant Maze itself poses a complex exploration problem due to sparse reward feedback. We make this problem even more complex by changing the success criteria. In this case, ASPiRe will carry three primitives: push, navigation, and avoid.

**Robotic Manipulation.** We further test ASPiRe's ability in complex robotic manipulation task in robosuite environment [49]. The task is to control a Panda robot arm to grasp the box (in red) without colliding with the barrier (in green) placed in the middle of the desk (See sub-figure 1 in Fig. 5).The location of the box and barrier are randomly generated at the beginning of the episode. ASPiRe will carry two primitive skill priors: grasp and avoid.

See appendix A.4 for details on environment, data collection process and training.

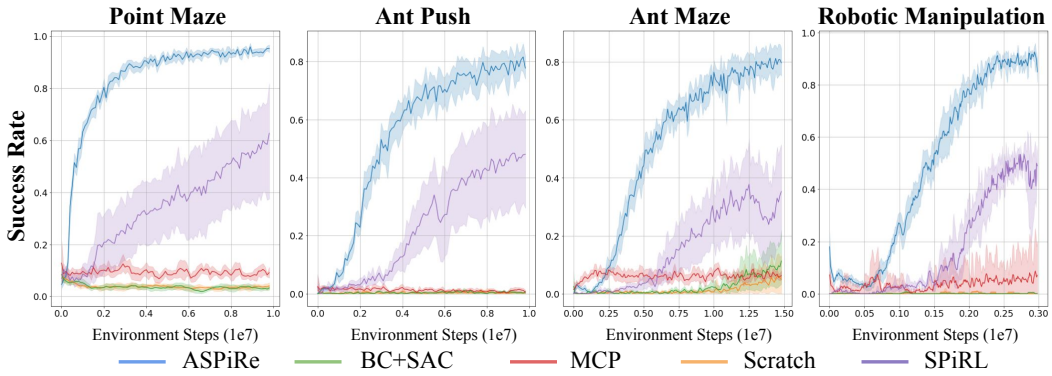


Figure 2: **Comparative Experiment.** Learning curves of our method and comparative methods for downstream tasks. ASPiRe significantly improves the learning efficiency and performance on all challenging tasks. The shaded area is the standard deviation over eight seeds.

**Alternative approaches.** We compare ASPiRe with several prior works for downstream task learning performance. To fairly evaluate the performance, we let all methods (besides SPiRL, as the method learns its own skill embedding) act on skill embedding space $\mathcal{Z}$ learned by the ASPiRe.

- **Scratch.** Train a Soft Actor-Critic (SAC, [50]) agent from scratch. This comparison tests the benefit of using the learned skill priors to guide downstream task learning.
- **Behavioral Cloning with finetuning.** A behavior cloning policy is first trained with offline data and then finetuned using SAC for the downstream tasks.
- **MCP**[51]**.** The method composes primitive policies through multiplicative composition of Gaussians with a learnable weighting function. MCP then samples the actions from the resulting multiplicative composite distribution. For a fair evaluation, MCP is directly given the same set of primitives that ASPiRe is equipped with. This comparison tests the benefit of treating composite skills as prior, instead of directly executing them.
- **SPiRL** [2]**.** Guide the policy learning with a single skill prior. We train the skill prior by combining all primitive datasets into one and learning a single prior from it. This comparison tests the benefit of guiding the learning policy with multiple primitive skill priors.

### 4.2 Experiment results

As shown in Fig. 2, our experiments demonstrate that ASPiRe is able to learn efficiently and converge to high success rates in all challenging environments.

**Benefits of multiple primitive skill priors.** We first compare ASPiRe with methods without skill prior guiding, namely training from scratch and behavior cloning with finetuning. They fail to solve

all three tasks (Fig. 2). This confirms the finding in Pertsch et al. [2] that skill prior is beneficial for learning complex and long-horizon tasks. Next, we compare our method with SPiRL [2], which regularizes the learned policy with a single skill prior extracted from the aggregated dataset. By leveraging the skill prior, it can achieve more efficient exploration but still relatively low learning speed and final success rates compared to ASPiRe (Fig. 2). We hypothesize that the performance gap between ASPiRe and SPiRL comes from the prior they deploy to guide the policy learning. We find that SPiRL learns an "average" behavior over all primitives. It prevents SPiRL from guiding states where the specific primitive skills are required. In the case of ASPiRe, it can construct an adaptive skill prior from all primitive skills priors. Therefore, the learned policy can receive sufficient guidance to explore and learn the tasks.

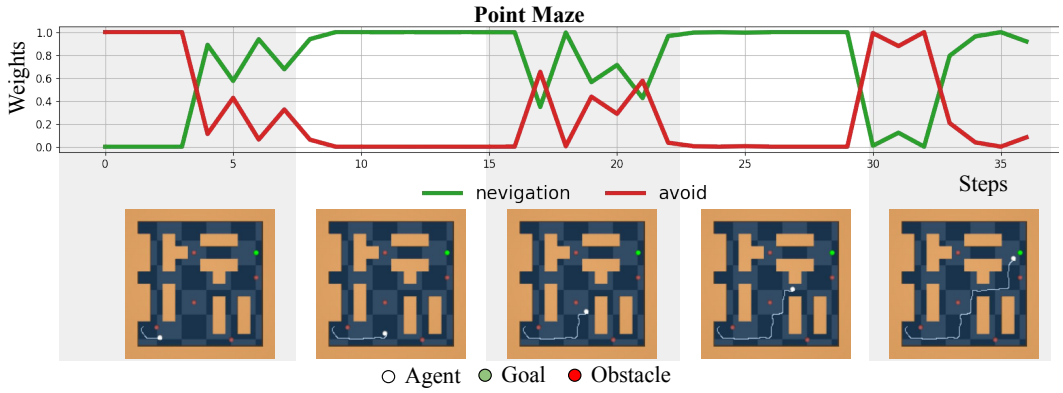


Figure 3: **Weights inference over one episode (Point Maze)**. ASPiRe is able to compose the navigation prior and avoid prior concurrently (grey phases) when the obstacle is observed and only activate navigation prior when it is not (white phases).

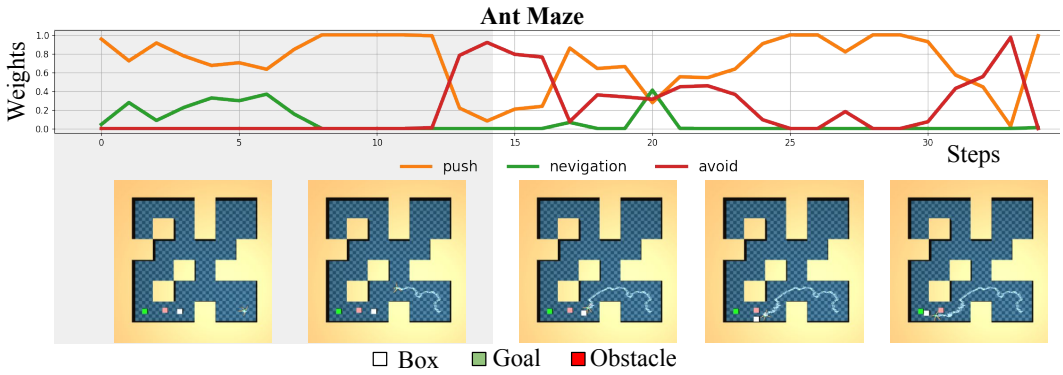


Figure 4: **Weights inference over one episode (Ant Maze)** In the grey phase, ASPiRe is able to compose the navigation prior and push prior concurrently to traverse the maze and reach the box. In the white phase, the model composes the push and avoid prior to push the box to the goal while avoiding the obstacle along the way.

**Benefits of using composite skills as regularization.** We compare ASPiRe with MCP, which directly executes the composite skills as a policy with a learnable weight. As shown in Figure 2, MCP struggles to learn a meaningful policy for all three tasks. This finding demonstrates that using composite skills as regularization provides the necessary flexibility for ASPiRe to learn the policies that can specifically solve the downstream task. We find the final learned policy has lower entropy than the skill priors in most cases. In fact, skill priors provide guidance on the valid skills to explore the environment, but the learned policy needs to further search in this valid skill space to solve the specific problem. We hypothesize that the resulting policy from MCP might be overly broad as the multiplicative composition of Gaussian inherent the relative high entropy from the primitives.

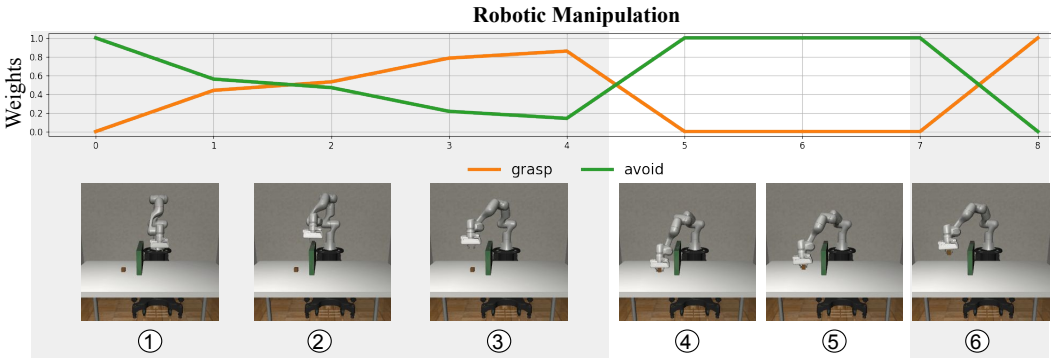


Figure 5: **Weights inference over one episode (Robotic Manipulation)** In the first grey phase (sub-figure 1, 2, 3), ASPiRe is able to compose the grasp prior and avoid prior concurrently to lift the robot arm to avoid the barrier while approach to the box. In the white phase, notice that the robot arm is initially very close to the barrier (sub-figure 4). To avoid any potential collision between the robot arm and the barrier, ASPiRe activates the avoid prior to move the robot arm away from the barrier (sub-figure5). In the end, ASPiRe activates the grasp prior to grasp the box.

**Learning optimal weights.** To demonstrate ASPiRe's ability to learn optimal weights, we visualize the weights generated by the weighting function for the primitive skills priors at different states for point maze, ant maze and robotic manipulation environments. From Fig. 3, 4 and 5, we find that ASPiRe can compose the primitive skill priors both sequentially and concurrently throughout the tasks. More importantly, ASPiRe regularizes the learned policy with relevant priors, e.g., activates the avoid primitive when obstacles are near the agent. See appendix A.1 for more trajectories.

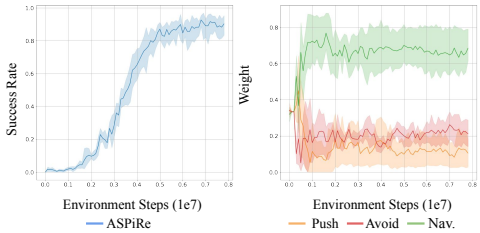


Figure 6: **Training with unrelated primitive.** Left: Learning curve. Right: Average weights of skill priors, where we can observe that ASPiRe assigns a lower weight for unrelated skill prior (e.g., push).

**Training with irrelevant primitive.** To test ASPiRe's ability in handling irrelevant primitive skill prior, we modify the task of the Ant Maze environment, where the ant needs to reach the goal while avoiding the obstacle along the way without pushing the box. We place the box in the lower right corner of the maze, which is not on the path to the goal. In this case, the push primitive is irrelevant, and regularizing with it will yield low learning efficiency. From Fig. 6, we find that ASPiRe still achieves good performance even with irrelevant primitive. We plot average weights over the states during the training. ASPiRe weights the navigation primitive heavily and assigns a low weight for the irrelevant primitive – push. The weight for avoid is also relatively low since there are fewer states where avoiding is needed (i.e., only when the obstacle is in view).

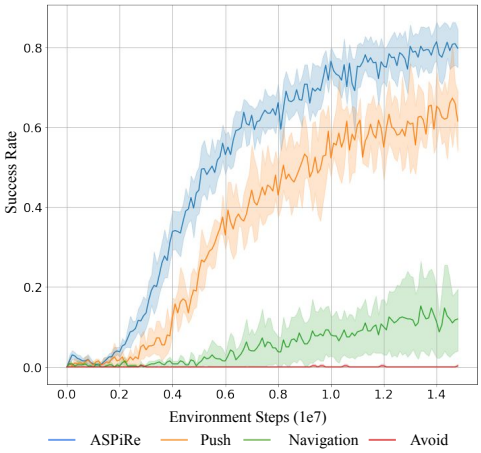


Figure 7: **Guiding with composite or single primitive.** Learning curves in Ant Maze.

**Guiding with composite or single primitive.** In this experiment, we test the advantage of using composite skill prior to guiding policy learning compared to guiding with a single primitive skill prior. To answer the question, we guide the policy learning with one of the primitive skill priors (i.e., push, navigation and avoid respectively) in ant maze environment. From Fig. 7, we find that the learned policy guided with only the avoid primitive or navigation primitive fails to learn the task. Guided with the push primitive achieves relatively good performance but fails to complete the setting requiring traverse in the maze. This experiment suggests that composing multiple skill priors is beneficial for the complex multi-task downstream tasks.

**Limitation and future work.** When extracting primitive skill priors, our algorithm requires separated and labeled datasets, each containing a specialized skill. However, such dataset may not always be easy to generate, and manually separating or defining different primitive skill priors can be challenging and sometimes ambiguous. Therefore, an interesting future direction will be allowing the system to process unlabeled dataset and automatically extract a set of distinct primitive skill priors.

## 5 Conclusion

We present ASPiRe, an approach to transfer a spectrum of skills from offline data to accelerate the learning of unseen downstream tasks. We propose to guide the policy learning with multiple primitive skill priors and transfer the diverse skills to RL via weighed KL divergence. We model an Adaptive Weight Module to learn optimal weights assignment and construct an adaptive skill prior. We evaluate ASPiRe in challenging, sparse reward environments and demonstrate the benefits of guiding the learning with multiple skill priors. We also show that ASPiRe is able to select the most relevant skill priors even when unrelated skill priors are presented.

## Acknowledgement

The Authors would like to thank Nelson Vadori, Sumitra Ganesh, Huy Ha, Cheng Chi, Samir Gadre, Zhenjia Xu, and Zhanpeng He on valuable feedback and support. Mengda Xu's work is supported by JPMorgan Chase & Co. This paper was prepared for information purposes in part by the Artificial Intelligence Research group of JPMorgan Chase & Co and its affiliates ("JP Morgan"), and is not a product of the Research Department of JP Morgan. JP Morgan makes no representation and warranty whatsoever and disclaims all liability, for the completeness, accuracy or reliability of the information contained herein. This document is not intended as investment research or investment advice, or a recommendation, offer or solicitation for the purchase or sale of any security, financial instrument, financial product or service, or to be used in any way for evaluating the merits of participating in any transaction, and shall not constitute a solicitation under any jurisdiction or to any person, if such solicitation under such jurisdiction or to such person would be unlawful.

## References

[1] Anurag Ajay, Aviral Kumar, Pulkit Agrawal, Sergey Levine, and Ofir Nachum. Opal: Offline primitive discovery for accelerating offline reinforcement learning. In *International Conference on Learning Representations*, 2020.

[2] Karl Pertsch, Youngwoon Lee, and Joseph J. Lim. Accelerating reinforcement learning with learned skill priors. In *Conference on Robot Learning (CoRL)*, 2020.

[3] Avi Singh, Huihan Liu, Gaoyue Zhou, Albert Yu, Nicholas Rhinehart, and Sergey Levine. Parrot: Data-driven behavioral priors for reinforcement learning. In *International Conference on Learning Representations*, 2020.

[4] Sergey Levine, Aviral Kumar, George Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *CoRR*, abs/2005.01643, 2020. URL `https://arxiv.org/abs/2005.01643`.

[5] Aviral Kumar, Justin Fu, G. Tucker, and Sergey Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. In *NeurIPS*, 2019.

[6] Yifan Wu, George Tucker, and Ofir Nachum. Behavior regularized offline reinforcement learning. *CoRR*, abs/1911.11361, 2019. URL `http://arxiv.org/abs/1911.11361`.

[7] Yao Liu, Adith Swaminathan, Alekh Agarwal, and Emma Brunskill. Off-policy policy gradient with state distribution correction. *CoRR*, abs/1904.08473, 2019. URL `http://arxiv.org/abs/1904.08473`.

[8] Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *ICML*, 2019.

[9] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. Conservative q-learning for offline reinforcement learning. *Advances in Neural Information Processing Systems*, 33: 1179–1191, 2020.

[10] Noah Siegel, Jost Tobias Springenberg, Felix Berkenkamp, Abbas Abdolmaleki, Michael Neunert, Thomas Lampe, Roland Hafner, Nicolas Heess, and Martin Riedmiller. Keep doing what worked: Behavior modelling priors for offline reinforcement learning. In *International Conference on Learning Representations*, 2019.

[11] Ziyu Wang, Alexander Novikov, Konrad Żołna, Jost Tobias Springenberg, Scott Reed, Bobak Shahriari, Noah Siegel, Josh Merel, Caglar Gulcehre, Nicolas Heess, et al. Critic regularized regression. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, pages 7768–7778, 2020.

[12] Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. Striving for simplicity in off-policy deep reinforcement learning. *CoRR*, abs/1907.04543, 2019. URL `http://arxiv.org/abs/1907.04543`.

[13] Natasha Jaques, Asma Ghandeharioun, Judy Hanwen Shen, Craig Ferguson, Àgata Lapedriza, Noah Jones, Shixiang Gu, and Rosalind W. Picard. Way off-policy batch deep reinforcement learning of implicit human preferences in dialog. *CoRR*, abs/1907.00456, 2019. URL `http://arxiv.org/abs/1907.00456`.

[14] Rahul Kidambi, Aravind Rajeswaran, Praneeth Netrapalli, and Thorsten Joachims. Morel: Model-based offline reinforcement learning. *Advances in neural information processing systems*, 33:21810–21823, 2020.

[15] Tianhe Yu, Garrett Thomas, Lantao Yu, Stefano Ermon, James Y Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. Mopo: Model-based offline policy optimization. *Advances in Neural Information Processing Systems*, 33:14129–14142, 2020.

[16] Arthur Argenson and Gabriel Dulac-Arnold. Model-based offline planning. In *International Conference on Learning Representations*, 2020.

[17] Tatsuya Matsushima, Hiroki Furuta, Yutaka Matsuo, Ofir Nachum, and Shixiang Gu. Deployment-efficient reinforcement learning via model-based offline optimization. In *International Conference on Learning Representations*, 2020.

[18] Catherine Cang, Aravind Rajeswaran, Pieter Abbeel, and Michael Laskin. Behavioral priors and dynamics models: Improving performance and domain transfer in offline rl. In *Deep RL Workshop NeurIPS 2021*, 2021.

[19] Ashvin Nair, Murtaza Dalal, Abhishek Gupta, and Sergey Levine. Accelerating online reinforcement learning with offline datasets. *CoRR*, abs/2006.09359, 2020. URL `https://arxiv.org/abs/2006.09359`.

[20] Dushyant Rao, Fereshteh Sadeghi, Leonard Hasenclever, Markus Wulfmeier, Martina Zambelli, Giulia Vezzani, Dhruva Tirumala, Yusuf Aytar, Josh Merel, Nicolas Heess, et al. Learning transferable motor skills with hierarchical latent mixture policies. In *International Conference on Learning Representations*, 2021.

[21] Fernando Fernández and Manuela Veloso. Probabilistic policy reuse in a reinforcement learning agent. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '06, page 720–727, New York, NY, USA, 2006. Association for Computing Machinery. ISBN 1595933034. doi: 10.1145/1160633.1160762. URL `https://doi.org/10.1145/1160633.1160762`.

[22] Brett Browning, James Bruce, Michael Bowling, and Manuela Veloso. Stp: Skills, tactics, and plays for multi-robot control in adversarial environments. *Proceedings of The Institution of Mechanical Engineers Part I-journal of Systems and Control Engineering - PROC INST MECH ENG I-J SYST C*, 219, 12 2004. doi: 10.1243/095965105X9470.

[23] Tuomas Haarnoja, Vitchyr Pong, Aurick Zhou, Murtaza Dalal, Pieter Abbeel, and Sergey Levine. Composable deep reinforcement learning for robotic manipulation. *CoRR*, abs/1803.06773, 2018. URL http://arxiv.org/abs/1803.06773.

[24] Jonathan J. Hunt, André Barreto, Timothy P. Lillicrap, and Nicolas Manfred Otto Heess. Composing entropic policies using divergence correction. In *ICML*, 2019.

[25] Benjamin Van Niekerk, Steven James, Adam Earle, and Benjamin Rosman. Composing value functions in reinforcement learning. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6401–6409. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/van-niekerk19a.html.

[26] André Barreto, Will Dabney, Rémi Munos, Jonathan J Hunt, Tom Schaul, Hado P van Hasselt, and David Silver. Successor features for transfer in reinforcement learning. *Advances in neural information processing systems*, 30, 2017.

[27] Andre Barreto, Diana Borsa, John Quan, Tom Schaul, David Silver, Matteo Hessel, Daniel Mankowitz, Augustin Zidek, and Remi Munos. Transfer in deep reinforcement learning using successor features and generalised policy improvement. In *International Conference on Machine Learning*, pages 501–510. PMLR, 2018.

[28] Diana Borsa, Andre Barreto, John Quan, Daniel J Mankowitz, Hado van Hasselt, Remi Munos, David Silver, and Tom Schaul. Universal successor features approximators. In *International Conference on Learning Representations*, 2018.

[29] André Barreto, Diana Borsa, Shaobo Hou, Gheorghe Comanici, Eser Aygün, Philippe Hamel, Daniel Toyama, Jonathan J. Hunt, Shibl Mourad, David Silver, and Doina Precup. The option keyboard: Combining skills in reinforcement learning. In *NeurIPS*, 2019.

[30] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture, 2016.

[31] Matthew Hausknecht and Peter Stone. Deep reinforcement learning in parameterized action space, 2016.

[32] Kevin Frans, Jonathan Ho, Xi Chen, Pieter Abbeel, and John Schulman. Meta learning shared hierarchies. In *International Conference on Learning Representations*, 2018.

[33] Josh Merel, Arun Ahuja, Vu Pham, Saran Tunyasuvunakool, Siqi Liu, Dhruva Tirumala, Nicolas Heess, and Greg Wayne. Hierarchical visuomotor control of humanoids. In *International Conference on Learning Representations*, 2018.

[34] Markus Wulfmeier, Abbas Abdolmaleki, Roland Hafner, Jost Tobias Springenberg, Michael Neunert, Tim Hertweck, Thomas Lampe, Noah Y. Siegel, Nicolas Heess, and Martin A. Riedmiller. Regularized hierarchical policies for compositional transfer in robotics. *CoRR*, abs/1906.11228, 2019. URL http://arxiv.org/abs/1906.11228.

[35] Nicolas Heess, Greg Wayne, Yuval Tassa, Timothy Lillicrap, Martin Riedmiller, and David Silver. Learning and transfer of modulated locomotor controllers, 2016.

[36] Carlos Florensa, Yan Duan, and Pieter Abbeel. Stochastic neural networks for hierarchical reinforcement learning. *CoRR*, abs/1704.03012, 2017. URL http://arxiv.org/abs/1704.03012.

[37] Josh Merel, Leonard Hasenclever, Alexandre Galashov, Arun Ahuja, Vu Pham, Greg Wayne, Yee Whye Teh, and Nicolas Heess. Neural probabilistic motor primitives for humanoid control. In *International Conference on Learning Representations*, 2018.

[38] Karol Hausman, Jost Tobias Springenberg, Ziyu Wang, Nicolas Heess, and Martin Riedmiller. Learning an embedding space for transferable robot skills. In *International Conference on Learning Representations*, 2018.

[39] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. In *International Conference on Learning Representations*, 2018.

[40] Tuomas Haarnoja, Kristian Hartikainen, P. Abbeel, and Sergey Levine. Latent space policies for hierarchical reinforcement learning. In *ICML*, 2018.

[41] Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87, 1991. doi: 10.1162/neco.1991.3.1.79.

[42] A. H. Qureshi, Jacob J. Johnson, Yuzhe Qin, Taylor Henderson, Byron Boots, and Michael C. Yip. Composing task-agnostic policies with deep reinforcement learning. In *ICLR*, 2020.

[43] Wei-Cheng Tseng, Jin-Siang Lin, Yao-Min Feng, and Min Sun. Toward robust long range policy transfer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 9958–9966, 2021.

[44] Jie Ren, Yewen Li, Zihan Ding, Wei Pan, and Hao Dong. Probabilistic mixture-of-experts for efficient deep reinforcement learning, 2021.

[45] Xiaofei Wang, Kimin Lee, Kourosh Hakhamaneshi, Pieter Abbeel, and Michael Laskin. Skill preferences: Learning to extract and execute robotic skills from human feedback. In *Conference on Robot Learning*, pages 1259–1268. PMLR, 2022.

[46] Yang Liu, Prajit Ramachandran, Qiang Liu, and Jian Peng. Stein variational policy gradient. In *33rd Conference on Uncertainty in Artificial Intelligence, UAI 2017*, 2017.

[47] Qiang Liu and Dilin Wang. Stein variational gradient descent: A general purpose bayesian inference algorithm. *Advances in neural information processing systems*, 29, 2016.

[48] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning, 2020.

[49] Yuke Zhu, Josiah Wong, Ajay Mandlekar, and Roberto Martín-Martín. robosuite: A modular simulation framework and benchmark for robot learning. In *arXiv preprint arXiv:2009.12293*, 2020.

[50] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[51] Xue Bin Peng, Michael Chang, Grace H. Zhang, P. Abbeel, and Sergey Levine. Mcp: Learning composable hierarchical control with multiplicative compositional policies. In *NeurIPS*, 2019.

## Checklist

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] See Section 4
   (c) Did you discuss any potential negative societal impacts of your work? [No]
   (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] The code is in the supplement material. We provide the codes to regenerate the datasets and downstream task learning and all instructions are inside readme.md

(b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See appendix A.4

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] The shaded area in Fig. 2 represents the standard deviation across 8 seeds

(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 appendix A.4

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] We use the D4RL benchmark and cite in the section 4.

(b) Did you mention the license of the assets? [No]

(c) Did you include any new assets either in the supplemental material or as a URL? [No]

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [No]

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [No]

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 Appendix

## A.1 Hyperparameter Sensitivity Analysis

There are two hyperparameters that can potentially impact the online learning for the downstream task: a) The target KL divergence $\delta$, which is used to automatically tune the temperature parameters in Eq. 4 and b) The sample size $M$ which is used to estimate the critic value in Eq. 7. We investigate the impact of the sample size $M$ on both Point Maze and Ant Maze environment and the choice of the target KL divergence $\delta$.

We first investigate the impact of sample size M by setting $M = 1, 10, 20$. As shown in Fig. 8, We find that the sample size has almost no impact on the learning. We hypothesize two reasons: a) The large train batch size in our experiment helps to estimate the critic value $Q(s, \omega)$ more stable and b) Given the input size of weight vector $\omega$ is small, it might be easy for the neural network to approximate its value via small sample size.

We further investigate the impact of the target KL divergence $\delta$. As shown in Fig. 9, we find that target KL divergence significantly impacts the learning. Based on our experiments, we conclude that a) More complex tasks require higher target KL divergence. Notice that the target KL divergence imposes on Ant Maze is higher than the one on Point Maze. The optimal policy to solve complex tasks might be significantly different from the composite skill prior. Therefore, the policy needs more "space" to explore around the composite skill prior. b) Imposing too small target KL divergence can lead to downgraded performance. From Fig. 9, the ASPiRe's performance drops significantly when we set $\delta = 6$ in Point Maze and $\delta = 10$ in Ant Maze. This is because the policy will be forced to stay close to the composite skill prior and itself might not be able to solve such complex tasks. c) Imposing too big target KL divergence can lead to downgraded performance. We observe that the learning is not efficient when setting $\delta = 32$ in Point Maze and $\delta = 50$ in Ant maze. As target KL divergence increases, the learned policy will receive less guidance from the prior. Though the learning is sensitive to the choice of target KL divergence, we find that there might still be a range of KL divergences values leading to the same optimal performance. Notice the ASPiRe achieves the same good performance when setting $\delta = 12$ or $\delta = 18$ in the Point Maze. We can also observe the same behavior on Ant Maze.

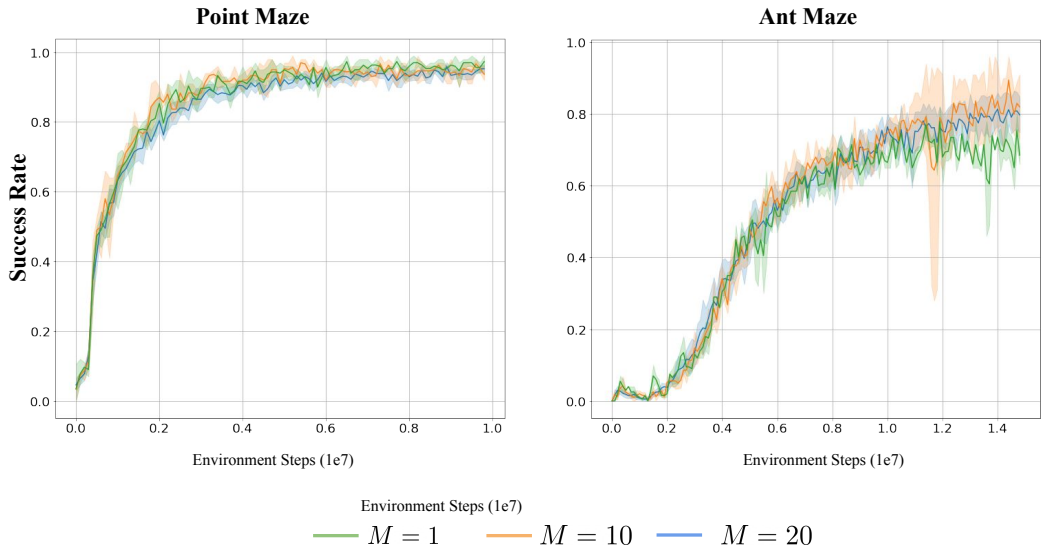


Figure 8: **Sensitivity towards sample size.** Learning curves of our method with different sample size $M$ in Point Maze and Ant Maze. The algorithm is not sensitive to this parameter.

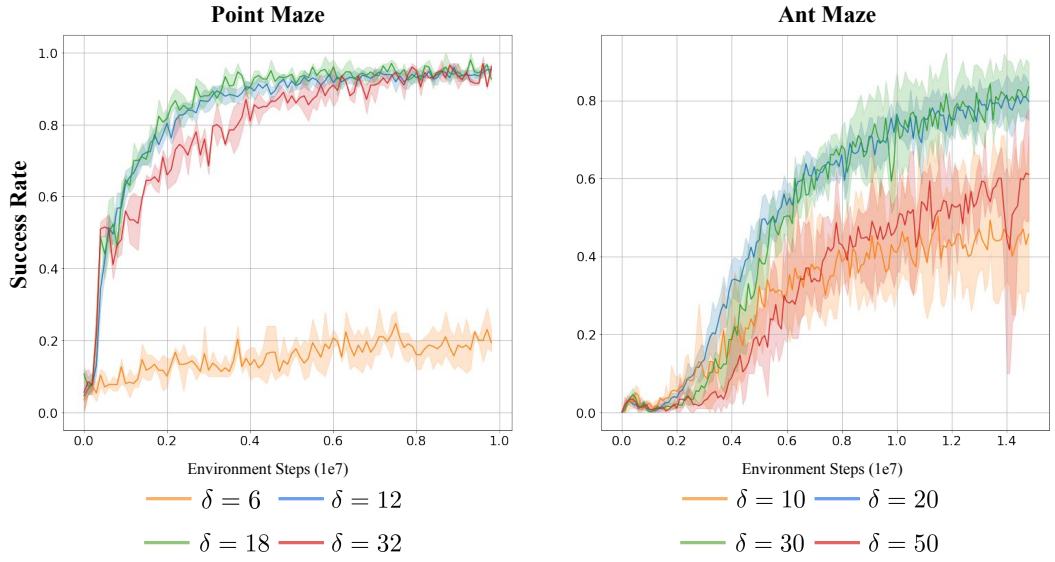


Figure 9: **Sensitivity towards target KL divergence.** Learning curves of our method with different target KL divergence $\delta$ in Point Maze and Ant Maze. More discussion in Sec A.1.

### A.2 Composite Skill Prior as Policy

We investigate the importance of using composite skill prior as regularization by comparing the ASPiRe with directly executing composite skill prior as a policy. This additional baseline is similar to the MCP. Both of them try to directly sample the action from composite distribution, but the difference is how they composite the distribution. One uses multiplicative gaussian (MCP) the other uses weighted KL divergence(ours).

From Fig. 10, we observe that using composite skill prior as a policy result in poor performance (similar performance as MCP). This confirms the importance of using composite skill prior as prior instead of policy.

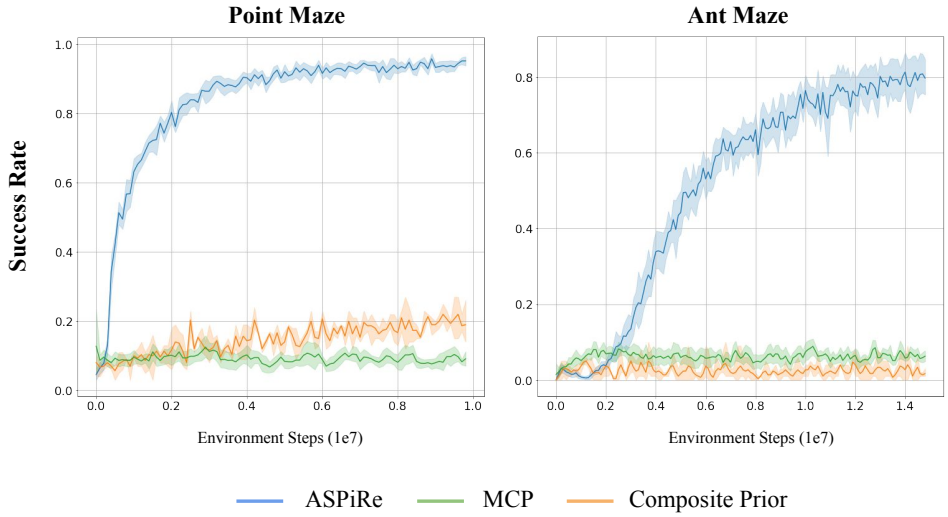


Figure 10: **Composite Skill Prior as Policy.** Learning curves of ASPiRe, MCP and Composite Skill Prior as policy in Point Maze and Ant Maze. Composite skill prior achieves similar performance as MCP which is much worse than ASPiRe.

### A.3 Uniform Weights

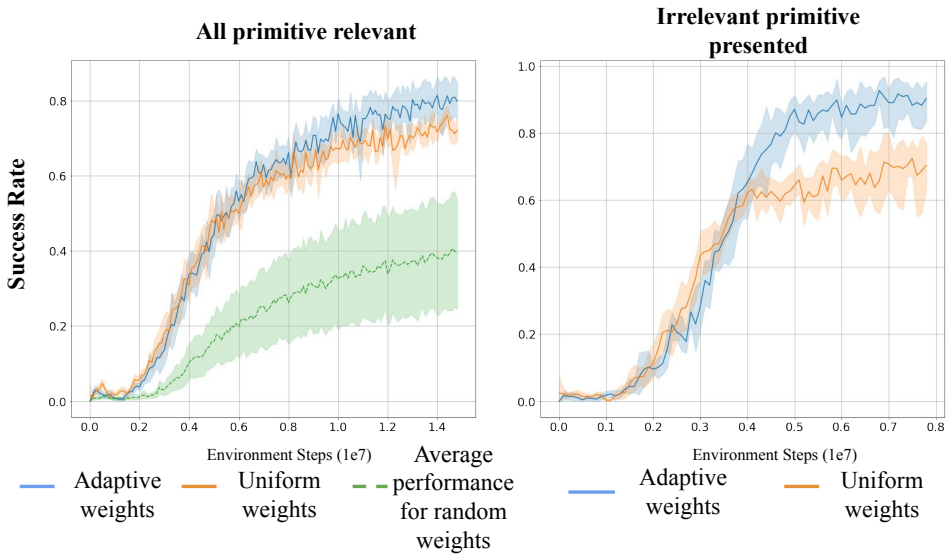


Figure 11: **Adaptive Weights vs Uniform Weights.** Learning curves of our method with Adaptive Weights and Uniform Weights in the setting (a) and (b). Notice that the tasks are different for this two settings. In the setting (a), The agent's goal is to traverse the maze while pushing the box to the goal position without hitting the obstacle. In the setting (b), the agent needs to reach the goal while avoiding the obstacle along the way without pushing the box.

We have tested the uniform weight in the Ant Maze domain. Specifically, we conduct the experiment in two different settings: a) all skill priors primitives are relevant to the downstream tasks b) not all skill priors are relevant to the downstream tasks.

In setting a) when all primitives are relevant, We observe that uniform weights also deliver good performance. If we compare it with many other random weights assignment (green line in Fig 11), uniform weights achieve a near-optimal assignment. The random weight performance is average of 6 different weight assignments. Regardless, our algorithm can still achieve the best performance among all weights assignments, showing that the system has enough flexibility to learn optimal policy.

In the setting b) not all skill priors are relevant to the downstream tasks. Our experiment shows that the adaptive weights assignment significantly outperforms the uniform weights. This suggests that our adaptive weight assignment is robust to the various downstream tasks.

### A.4 Implementation Details

#### A.4.1 Offline phase

The skill encoder is a one-layer LSTM with 128 hidden units, which receives $H = 10$ steps action sequence as input and outputs the parameters of the Gaussian posterior $\mathcal{N}(\mu_z, \sigma_z)$ on a 10 dimensional skill embedding. The skill decoder is also a one-layer LSTM with 128 hidden units, which reconstructs the latent embedding back to the H steps action sequence, i.e., skill. Each primitive skill prior is parameterized with a 6-layer MLP with 128 hidden units per layer. We use ReLU as our activation layer and apply batch normalization. The network outputs parameters of the Gaussian skill prior $\mathcal{N}(\mu_a, \sigma_a)$.

The models are optimized with Adam with learning rate 1e-4and batch size 64. In the offline learning phase, we first learn a shared low-dimensional skill embedding space $\mathcal{Z}$ from aggregated datasets $\{\mathcal{D}_i\}_{i=0}^{K}$. For each iteration, we select one of the primitive datasets $\mathcal{D}_i$ and sample the state-skill tuples from it to optimize the skill embedding and the corresponding primitive skill prior $p_a^i(z|s)$. Once the ELBO loss in section 3.2 converges, we freeze the skill encoder/decoder parameters and only update the parameters of primitive skill prior. This makes sure that primitive skill priors can be learned on a stationary skill embedding space. We set the regularization parameter $\beta =$ 1e-4 for all experiment domains.

#### A.4.2 Online phase

The learned policy $\pi_\theta(s)$ is parameterized with a 6-layer MLP with 128 hidden units per layer. The network outputs the parameters of a Gaussian distribution. We limit the action range of the policy between $[-2, ..., 2]$ by a tanh function. The weighting function $\omega_\sigma(s)$ is also parameterized with a 6-layer MLP with k-way softmax as the output layer. We use ReLU as our activation layer and apply batch normalization for both the learned policy and the weighting function. We model two critic networks and take the minimum value as Q-value estimation, which stabilizes the training. Each critic network is implemented as 4-layer MLP with 256 hidden units per layer. Skill prior generator $G_\sigma(z|s)$ is parameterized with a 3-layer MLP with 128 hidden units per layer. The network outputs the parameters of a Gaussian distribution.

We optimized the models with Adam and set the learning rate as 1e-4. The replay buffer capacity is 1e6, and the batch size is 256. We empirically find that setting the discount factor $\gamma$ as 0.97 can slightly stabilize the learning. The annealing coefficient $\beta$ starts with 1 and gradually decreases to 1e-3. We set the target divergence $\delta = 12$ for Point Maze, $\delta = 15$ for Ant Push, and $\delta = 20$ for Ant Maze. The temperature parameter $\alpha$ can be tuned automatically by minimizing the following:

$$\arg\min_{\alpha>0} \mathbb{E}\big[\alpha\delta - \alpha \sum_{i=1}^{K} \omega_i(s_t) D_{KL}(\pi(z_t|s_t), p_a^i(z_t|s_t))\big] \tag{9}$$

The prove can be found in **?** ]. We replace the single term KL divergence in the original formulation with weighted KL divergence. All models are trained on a single NVIDIA GPU.

### A.5 Environments and data collection

For downstream tasks learning (online phase), the state $s_t$ can be partitioned into two components:

$$s_t = s_t^{\mathrm{p}} + g_t$$

where $s_t^{\mathrm{p}}$ is the proprioceptive observation and $g_t$ is the downstream task-related observation. The states in each primitive dataset $\mathcal{D}_i$ always include the proprioceptive observation $s_t^{\mathrm{p}}$, but part of the downstream task-related observation might be absent. For example, during the learning of Point Maze, the proprioceptive observation $s_t^{\mathrm{p}}$ includes the agent's local view and velocity, and the task-related observation $g_t$ includes the positions of the obstacle, the goal, and the agent. However, in the navigation dataset, the whole task-related observation is absent. Suppose we extract the navigation primitive skill prior based on the proprioceptive observation only. In that case, the resulting prior cannot be directly applied to the downstream task as the state dimensions are different. To remedy this issue, we augment the states in offline data with random vectors that have the same length with the absent task-related observation. This augmentation will not impact the prior learning as the network will eventually ignore random vectors during the extracting process. Alternatively, instead

of augmentation, we can parse the states during downstream task learning and only feed the relevant states to primitive skill priors. We have experienced both ways and find the choice does not impact the performance. For the experiment result in this paper, we apply augmentation. This issue might be solved by a method like Attention. However, this is out of the scope of this paper, and we will leave it for future research.

### A.5.1 Point Maze

The navigation dataset is collected by running the planning algorithm (provided in D4RL benchmark) in multiple random generated maze layouts. The avoid dataset is collected by a simple heuristic algorithm, i.e., the agent heads in any direction that can prevent itself from colliding with the obstacle. The heading direction $\theta$ can be calculated as: $\theta = -\frac{x_t^{agent} - x_t^{obstacle}}{||x_t^{agent} - x_t^{obstacle}||} \pm \gamma$, where $\gamma$ is a random number sampled from $[-\frac{\pi}{4}, \frac{\pi}{4}]$, $x_t^{agent}$ is the position of the agent, and $x_t^{obstacle}$ is the position of the obstacle. The proprioceptive observation $s_t^{\text{p}}$ in primitive datasets includes the agent's local view and $v_t^{agent}$, which is the velocity of the agent.

For the downstream task, the obstacles, goal, and agent positions are randomly reset at the beginning of one episode. We set a minimum distance between the goal and the agent position as 5 to avoid trivial tasks to constrain the reset process. The point agent will receive a sparse positive reward when the agent reaches the goal and a sparse negative reward when the agent hits the obstacle. The episode will be immediately reset once the agent receives either a positive or a negative reward. The additional task-related state $g_t$ is represented as a 30-dimensional vector which includes the local observation (i.e., goal/obstacle if in view) and the goal position. The action is a 2-dimensional continuous space that controls the agent's velocity in x and y-direction.

### A.5.2 Ant Push

To collect the push and avoid dataset, we first train a reaching policy by SAC, which allows the ant to head towards a specific target. The reaching policy takes a vector input $(x_t^{goal} - x_t^{agent}, s_t^{\text{p}})$, where $x_t^{goal}$ is the goal position, $x_t^{agent}$ is the agent position, and proprioceptive observation $s_t^{\text{p}}$ is a 29-dimensional vector that represents the ant's joint state. The policy outputs an 8-dimensional vector to control the ant's joint. To collect the push dataset, we randomly place the ant at a distance of $[2, 8]$ from the box. We use the reaching policy to create push behavior by setting the goal as the center of the box. The avoid dataset is generated by the same heuristic policy in the Point Maze environment.

For the downstream task, the obstacle, goal, and agent positions are randomly reset at the beginning of one episode. We randomly initialize the goal position $x_0^{goal}$ at a distance of $[-4, 4]$ from the center $(0, 0)$ and the agent position $x_0^{agent}$ at a distance of $[-8, 8]$ from the center $(0, 0)$. The obstacle is initialized between the goal and the agent $x_0^{obstacle} = \frac{x_0^{goal} + x_0^{agent}}{2} + \epsilon$, where $\epsilon$ is sampled from Uniform(-1,1). Like Point Maze, we set a minimum initial distance between the goal and the agent position as 2 to avoid trivial cases. The agent will receive a sparse positive reward when the agent reaches the goal and a sparse negative reward when the agent hits the obstacle. The additional task-related state $g_t$ is a 4-dimensional vector representing the position difference between the agent and the goal $x_t^{goal} - x_t^{agent}$, and the position difference between the agent and the obstacle $x_t^{obstacle} - x_t^{agent}$. The action is an 8-dimensional continuous space.

### A.5.3 Ant Maze

We reuse the push and avoid dataset in Ant Push to extract the push and avoid primitive. Navigation primitive is extracted from the dataset (antmaze-medium-diverse) in D4RL benchmark. The proprioceptive observation $s_t^{\text{p}}$ is a 29-dimensional vector that represents the ant's joint state.

For the downstream task learning, we modify the Ant Maze environment from D4RL benchmark by adding an obstacle and a box in the maze. There are 6 pre-defined sub-tasks in the Ant Maze environment, and each of the sub-tasks specifies distinct obstacle, box, and goal positions. The agent will be given a task that is randomly selected from pre-defined sub-tasks at the beginning of every episode. The reward setting is the same as the one in Ant Push. However, the additional maze structure posts a hard exploration problem. The additional task-related state $g_t$ is a 6-dimensional

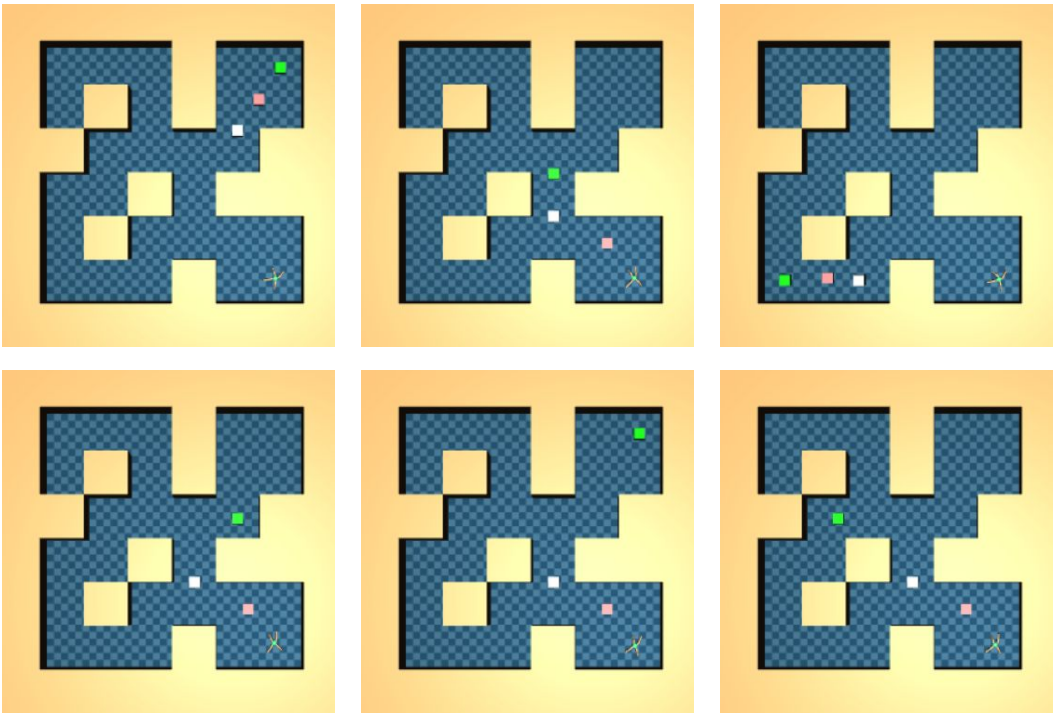

Figure 12: **Pre-defined sub-tasks in Ant Maze**

vector representing the position difference between the agent and the goal $x_t^{goal} - x_t^{agent}$, and the position difference between the agent and the obstacle $x_t^{obstacle} - x_t^{agent}$ and the agent position $x_t^{agent}$. The action is an 8-dimensional continuous space.

#### A.5.4 Robotic Manipulation

The task is to control a Panda robot arm to grasp the box without colliding with the barrier placed in the middle of the desk. The height of the barrier is set to be higher than the initial position of the robot arm's end effector. The location of the box and barrier are randomly generated at the beginning of the episode. The agent will receive a sparse positive reward once it grasps the box and lift it up without colliding with the barrier. ASPiRe will carry two primitive skill priors: grasp and avoid. The grasp data is generated by a hand-coded heuristic policy, which first moves the end effector horizontally to the top of the object and then moves the end effector down to grasp the object. The avoid data is also generated by a hand-coded heuristic policy, which moves the end effector up vertically once the barrier is observed. As the datasets are collected by heuristic policies, a part of trajectories might fail to complete the primitive tasks, i.e., grasp the box. Therefore, we only keep the trajectories that successfully complete the primitive tasks.