Accelerating Reinforcement Learning with Learned Skill Priors

Karl Pertsch **Youngwoon Lee** **Joseph J. Lim**

Department of Computer Science
University of Southern California
{pertsch,lee504,limjj}@usc.edu

Abstract

Intelligent agents rely heavily on prior experience when learning a new task, yet most modern reinforcement learning (RL) approaches learn every task from scratch. One approach for leveraging prior knowledge is to transfer *skills* learned on prior tasks to the new task. However, as the amount of prior experience increases, the number of transferable skills grows too, making it challenging to explore the full set of available skills during downstream learning. Yet, intuitively, not all skills should be explored with equal probability; instead information e.g., about the environment state can hint which skills are promising to explore. We propose to implement this intuition by learning a *prior over skills*. We propose a deep latent variable model that jointly learns an embedding space of skills and the skill prior from offline agent experience. We then extend common maximum-entropy RL approaches to incorporate skill priors to guide downstream learning. We validate our approach on complex navigation and robotic manipulation tasks and show that learned skill priors are essential for effective transfer of skills from rich datasets. For a more detailed version of the paper, videos and code, see `clvrai.com/spirl`.

1 Introduction

Intelligent agents are able to utilize a large pool of prior experience to efficiently learn how to solve new tasks [37]. In contrast, reinforcement learning (RL) agents typically learn each new task *from scratch*, without leveraging prior experience. On the other hand, there is an abundance of collected agent experience available in domains like autonomous driving [3], indoor navigation [24], or robotic manipulation [4, 2]. In this work, our aim is to devise a scalable approach for leveraging such unstructured experience to accelerate the learning of new downstream tasks.

One flexible way to utilize unstructured prior experience is by extracting *skills*, temporally extended actions that represent useful behaviors, which can be repurposed to solve downstream tasks. Prior work has learned skill libraries from data collected by humans [27, 22, 23, 29, 19] or by agents autonomously exploring the world [12, 30]. To solve a downstream task using the learned skills, these approaches train a high-level policy whose action space is the set of extracted skills. The dimensionality of this action space scales with the number of skills. The large skill libraries extracted from rich datasets can, somewhat paradoxically, lead to worse learning efficiency since the agent needs to collect large amounts of experience to perform the necessary exploration in skill space [14].

The key idea of this work is to learn a *prior over skills* along with the skill library to guide exploration in skill space and enable efficient downstream learning, even with large skill spaces. Intuitively, the prior over skills is not uniform: if the agent holds the handle of a cup, it is more promising to explore a pick-up skill than a sweeping skill (see Fig. 1). To implement this idea, we design a stochastic latent variable model that learns a continuous embedding space of skills and a prior distribution over these skills from unstructured agent experience. We then show how to naturally incorporate the learned

NeurIPS 2020 3rd Robot Learning Workshop: Grounding Machine Learning Development in the Real World.

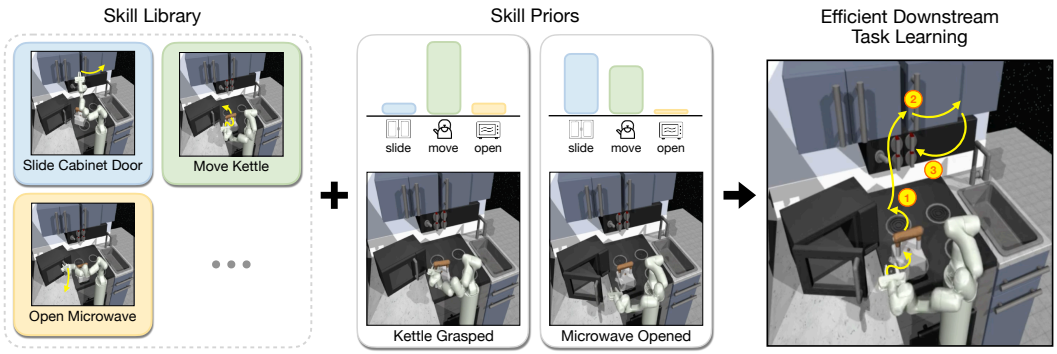

Figure 1: Intelligent agents can use a large library of acquired skills when learning new tasks. Instead of exploring skills uniformly, they can leverage *priors over skills* as guidance, based e.g., on the current environment state. Such priors capture which skills are promising to explore, like moving a kettle when it is already grasped, and which are less likely to lead to task success, like attempting to open an already opened microwave. In this work, we propose to jointly learn an embedding space of skills and a prior over skills from unstructured data to accelerate the learning of new tasks.

skill priors into maximum-entropy RL algorithms for efficient learning of downstream tasks. We validate our approach on complex, long-horizon navigation and robot manipulation tasks. We show that through the transfer of skills we can use unstructured experience for accelerated learning of new tasks and that learned skill priors are essential to scale to rich experience datasets.

In summary, the contributions of this work are threefold: (1) we design a model for jointly learning an embedding space of skills and a prior over skills from unstructured data, (2) we extend maximum-entropy RL to incorporate learned skill priors for efficient task learning, and (3) we show that learned skill priors accelerate learning across three simulated navigation and robot manipulation tasks.

2 Related Work

The problem of inter-task transfer has been studied for a long time in the RL community [34]. The idea of **transferring skills between tasks** dates back at least to the SKILLS [35] and *PolicyBlocks* [25] algorithms. Learned skills can be represented as sub-policies in the form of options [33, 1], as subgoal setter and reacher functions [9, 21] or discrete primitive libraries [27, 17]. Recently, a number of works have explored the embedding of skills into a continuous *skill space* via stochastic latent variable models [12, 22, 16, 23, 29, 36, 19]. When using powerful latent variable models, these approaches are able to represent a very large number of skills in a compact embedding space. However, the exploration of such a rich skill embedding space can be challenging, leading to inefficient downstream task learning [14]. Our work introduces a learned skill prior to guide the exploration of the skill embedding space, enabling efficient learning on rich skill spaces.

Learned behavior priors are commonly used to guide task learning in offline RL approaches [7, 13, 38] in order to avoid value overestimation for actions outside of the training data distribution. Recently, action priors have been used to leverage offline experience for learning downstream tasks [31]. Crucially, our approach learns priors over *temporally extended actions*, i.e. skills, allowing it to scale to complex, long-horizon downstream tasks.

3 Approach

Our goal is to leverage skills extracted from large, unstructured datasets to accelerate the learning of new downstream tasks. Scaling skill transfer to large datasets is challenging, since learning the downstream task requires picking the appropriate skills from an increasingly large library of extracted skills. We assume access to a dataset $\mathcal{D}$ of pre-recorded agent experience in the form of state-action trajectories $\tau_i = \{(s_0, a_0), \ldots, (s_{T_i}, a_{T_i})\}$. This data can be collected using previously trained agents across a diverse set of tasks [6, 8], through agents autonomously exploring their environment [12, 30],

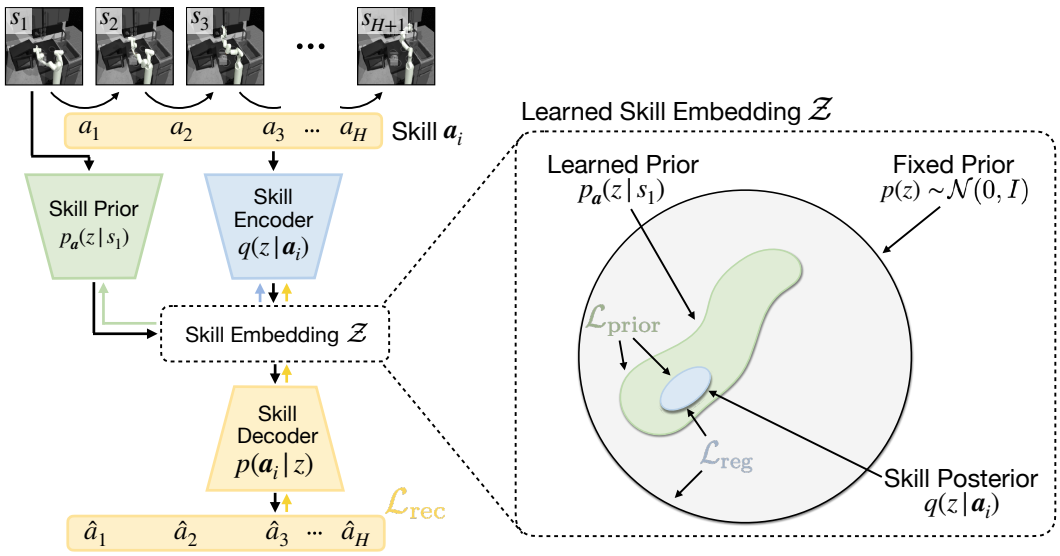

Figure 2: Deep latent variable model for joint learning of skill embedding and skill prior. Given a state-action trajectory from the dataset, the skill encoder maps the action sequence to a posterior distribution $q(z|\boldsymbol{a}_i)$ over latent skill embeddings. The action trajectory gets reconstructed by passing a sample from the posterior through the skill decoder. The skill prior maps the current environment state to a prior distribution $p_{\boldsymbol{a}}(z|s_1)$ over skill embeddings. Colorful arrows indicate the propagation of gradients from reconstruction, regularization and prior training objectives.

via human teleoperation [28, 9, 20, 19] or any combination of these. Crucially, we aim to leverage *unstructured* data that does not have annotations of tasks or sub-skills and does not contain reward information to allow for scalable data collection on real world systems.

3.1 Learning Continuous Skill Embeddings and Skill Priors

We define a skill $\boldsymbol{a}_i$ as a sequence of actions $\{a_t^i, \dots, a_{t+H-1}^i\}$ with fixed horizon H. Using fixed-length skills allows for scalable skill learning and has proven to be effective in prior works [22, 23, 9, 36, 21, 5]. To learn a low-dimensional skill embedding space $\mathcal{Z}$, we train a stochastic latent variable model $p(\boldsymbol{a}_i|z)$ of skills using the offline dataset (see Fig. 2). We randomly sample H-step trajectories from the training sequences and maximize the following evidence lower bound (ELBO): $\log p(\boldsymbol{a}_i) \geq \mathbb{E}_q\Big[\underbrace{\log p(\boldsymbol{a}_i|z)}_{\text{reconstruction}} - \beta\big(\underbrace{\log q(z|\boldsymbol{a}_i) - \log p(z)}_{\text{regularization}}\big)\Big]$. We optimize this objective using amortized variational inference with a learned inference model $q(z|\boldsymbol{a}_i)$ [15, 26].

Our aim is to learn a prior over skills along with the skill embedding model. We therefore introduce another component in our model: the skill prior $p_{\boldsymbol{a}}(z|\cdot)$. The input to this skill prior can be adjusted to the environment and task at hand; in this work we focus on learning a state-conditioned skill prior $p_{\boldsymbol{a}}(z|s_t)$. Intuitively, the current state should provide a strong prior over which skills are promising to explore and, maybe more importantly, which skills should not be explored in the current situation (see Fig. 1). To train the skill prior we minimize the Kullback-Leibler divergence between the predicted prior and the inferred skill posterior: $\mathbb{E}_{(s,\boldsymbol{a}_i)\sim\mathcal{D}} D_{\text{KL}}\big(q(z|\boldsymbol{a}_i), p_{\boldsymbol{a}}(z|s_t)\big)$.

3.2 Skill Prior Regularized Reinforcement Learning

To use the learned skill embedding and skill prior for efficient downstream task learning, we employ a hierarchical policy learning scheme by using the learned skill embedding space as the action space of a high-level policy. Concretely, instead of learning a policy over actions $a \in \mathcal{A}$ we learn a policy over skill embeddings $\pi_\theta(z|s_t)$. We execute the actions $\{a_t^i, \dots, a_{t+H-1}^i\} \sim p(\boldsymbol{a}_i|z)$ for H steps before sampling the next skill from the high-level policy.

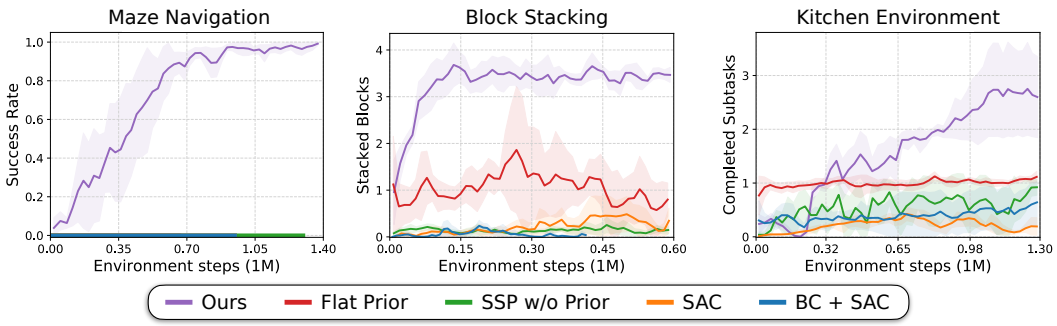

Figure 3: Downstream task learning curves for our method and all comparisons. Both, learned skill embeddings and skill priors are essential for downstream task performance: single-action priors without temporal abstraction (**Flat Prior**) and learned skills without skill prior (**SSP w/o Prior**) fail to converge to good performance. Shaded areas represent standard deviation across three seeds.

We can use standard model-free RL to optimize the high-level policy. In this work we propose to use the learned skill prior to guide the learning process. We extend maximum entropy RL [39, 18] approaches by replacing the entropy term that gets added to the reward, representing the divergence from an *uniform* action prior, with a divergence term towards our learned skill prior, leading to the following objective: $J(\theta) = \mathbb{E}_\pi\left[\sum_{t=1}^{T} \tilde{r}(s_t, z_t) - \alpha D_{\text{KL}}\big(\pi(z_t|s_t), p_{\boldsymbol{a}}(z_t|s_t)\big)\right]$.

We can modify the state-of-the-art maximum-entropy RL algorithms, such as Soft Actor-Critic (SAC, [10, 11]) to optimize this objective. We summarize our skill-prior regularized SAC approach in algorithm 1 in the appendix with changes to SAC marked in red.

4 Experiments

We test our approach on a maze navigation task with sparse rewards and two challenging robotic manipulation tasks: block stacking and long-horizon manipulations with a 7DoF robotic arm in a simulated kitchen environment (see Fig. 4). We compare to conventional model free RL (SAC, [10], w/ and w/o behavioral cloning initialization); learning priors over primitive actions not skills ("Flat Prior", similar to [31]); leveraging transferred skills without guidance via a learned prior ("w/o prior", representative of [22, 16, 29]). More details on environments and comparisons are in appendix, section D. Detailed ablation studies are in section E.

Maze Navigation We first evaluate our approach on the sparse-reward maze navigation task. In Fig. 3 (left) we show that only our method is able to learn a goal-reaching policy. To better understand this result, we compare the exploration behavior of our approach and the baselines in Fig. 6: only our approach is able to explore large parts of the maze. Random exploration in skill space does not lead to good exploration behavior since the number of possible skills is too large and targeted sampling is required to e.g. navigate through doorways. The comparison to single-step action priors ("Flat Prior") shows that temporal abstraction is beneficial for coherent exploration. We further show that a single learned prior can be reused for learning to reach a variety of goals in the maze in appendix, section F.

Robotic Manipulation For both robotic manipulation environments we find that using learned skill embeddings together with the extracted skill prior is essential to solve the task (see Fig. 3, middle and right; appendix Fig. 7 for qualitative policy rollouts). In contrast, using non-hierarchical action priors ("Flat Prior") leads to performance similar to behavioral cloning of the training dataset, but fails to solve longer-horizon tasks. The approach leveraging the learned skill space without guidance from the skill prior ("SSP w/o Prior") only rarely stacks blocks or successfully manipulates objects in the kitchen environment. Due to the large number of extracted skills from the rich training datasets, random exploration in skill space does not lead to efficient learning, underlining the importance of learned skill priors for scaling skill transfer to large datasets.

References

[1] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *AAAI*, 2017.

[2] Serkan Cabi, Sergio Gomez Colmenarejo, Alexander Novikov, Ksenia Konyushkova, Scott Reed, Rae Jeong, Konrad Zolna, Yusuf Aytar, David Budden, Mel Vecerik, Oleg Sushkov, David Barker, Jonathan Scholz, Misha Denil, Nando de Freitas, and Ziyu Wang. Scaling data-driven robotics with reward sketching and batch reinforcement learning. *RSS*, 2019.

[3] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. *preprint arXiv:1903.11027*, 2019.

[4] Sudeep Dasari, Frederik Ebert, Stephen Tian, Suraj Nair, Bernadette Bucher, Karl Schmeckpeper, Siddharth Singh, Sergey Levine, and Chelsea Finn. Robonet: Large-scale multi-robot learning. *CoRL*, 2019.

[5] Kuan Fang, Yuke Zhu, Animesh Garg, Silvio Savarese, and Li Fei-Fei. Dynamics learning with cascaded variational inference for multi-step manipulation. *CoRL 2019*, 2019.

[6] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

[7] Scott Fujimoto, David Meger, and Doina Precup. Off-policy deep reinforcement learning without exploration. In *International Conference on Machine Learning*, pages 2052–2062, 2019.

[8] Caglar Gulcehre, Ziyu Wang, Alexander Novikov, Tom Le Paine, Sergio Gómez Colmenarejo, Konrad Zolna, Rishabh Agarwal, Josh Merel, Daniel Mankowitz, Cosmin Paduraru, et al. Rl unplugged: Benchmarks for offline reinforcement learning. *arXiv preprint arXiv:2006.13888*, 2020.

[9] Abhishek Gupta, Vikash Kumar, Corey Lynch, Sergey Levine, and Karol Hausman. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. *CoRL*, 2019.

[10] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *ICML*, 2018.

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

[12] 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.

[13] Natasha Jaques, Asma Ghandeharioun, Judy Hanwen Shen, Craig Ferguson, Agata Lapedriza, Noah Jones, Shixiang Gu, and Rosalind Picard. Way off-policy batch deep reinforcement learning of implicit human preferences in dialog. *arXiv preprint arXiv:1907.00456*, 2019.

[14] Nicholas K Jong, Todd Hester, and Peter Stone. The utility of temporal abstraction in reinforcement learning. In *AAMAS (1)*, pages 299–306. Citeseer, 2008.

[15] Diederik P Kingma and Max Welling. Auto-encoding variational Bayes. In *ICLR*, 2014.

[16] Thomas Kipf, Yujia Li, Hanjun Dai, Vinicius Zambaldi, Edward Grefenstette, Pushmeet Kohli, and Peter Battaglia. Compositional imitation learning: Explaining and executing one task at a time. *ICML*, 2019.

[17] Youngwoon Lee, Shao-Hua Sun, Sriram Somasundaram, Edward S Hu, and Joseph J Lim. Composing complex skills by learning transition policies. In *International Conference on Learning Representations*, 2018.

[18] Sergey Levine. Reinforcement learning and control as probabilistic inference: Tutorial and review. *arXiv preprint arXiv:1805.00909*, 2018.

[19] Corey Lynch, Mohi Khansari, Ted Xiao, Vikash Kumar, Jonathan Tompson, Sergey Levine, and Pierre Sermanet. Learning latent plans from play. In *Conference on Robot Learning*, pages 1113–1132, 2020.

[20] Ajay Mandlekar, Yuke Zhu, Animesh Garg, Jonathan Booher, Max Spero, Albert Tung, Julian Gao, John Emmons, Anchit Gupta, Emre Orbay, Silvio Savarese, and Li Fei-Fei. Roboturk: A crowdsourcing platform for robotic skill learning through imitation. In *Conference on Robot Learning*, 2018.

[21] Ajay Mandlekar, Fabio Ramos, Byron Boots, Li Fei-Fei, Animesh Garg, and Dieter Fox. Iris: Implicit reinforcement without interaction at scale for learning control from offline robot manipulation data. *ICRA*, 2020.

[22] 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. *ICLR*, 2019.

[23] Josh Merel, Saran Tunyasuvunakool, Arun Ahuja, Yuval Tassa, Leonard Hasenclever, Vu Pham, Tom Erez, Greg Wayne, and Nicolas Heess. Catch & carry: Reusable neural controllers for vision-guided whole-body tasks. *ACM. Trans. Graph.*, 2020.

[24] Kaichun Mo, Haoxiang Li, Zhe Lin, and Joon-Young Lee. The AdobeIndoorNav Dataset: Towards deep reinforcement learning based real-world indoor robot visual navigation. *preprint arXiv:1802.08824*, 2018.

[25] Marc Pickett and Andrew G Barto. Policyblocks: An algorithm for creating useful macro-actions in reinforcement learning. In *ICML*, volume 19, pages 506–513, 2002.

[26] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *ICML*, 2014.

[27] Stefan Schaal. *Dynamic Movement Primitives - A Framework for Motor Control in Humans and Humanoid Robotics*. Springer Tokyo, 2006.

[28] Stefan Schaal, Jan Peters, Jun Nakanishi, and Auke Ijspeert. Learning movement primitives. In Paolo Dario and Raja Chatila, editors, *Robotics Research*. Springer Berlin Heidelberg, 2005.

[29] Tanmay Shankar, Shubham Tulsiani, Lerrel Pinto, and Abhinav Gupta. Discovering motor programs by recomposing demonstrations. In *International Conference on Learning Representations*, 2019.

[30] Archit Sharma, Shixiang Gu, Sergey Levine, Vikash Kumar, and Karol Hausman. Dynamics-aware unsupervised discovery of skills. *ICLR*, 2020.

[31] Noah Y Siegel, Jost Tobias Springenberg, Felix Berkenkamp, Abbas Abdolmaleki, Michael Neunert, Thomas Lampe, Roland Hafner, and Martin Riedmiller. Keep doing what worked: Behavioral modelling priors for offline reinforcement learning. *ICLR*, 2020.

[32] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[33] Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2): 181–211, 1999.

[34] Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(7), 2009.

[35] Sebastian Thrun and Anton Schwartz. Finding structure in reinforcement learning. In *NIPS*, 1995.

[36] William Whitney, Rajat Agarwal, Kyunghyun Cho, and Abhinav Gupta. Dynamics-aware embeddings. *ICLR*, 2020.

[37] Robert S Woodworth and EL Thorndike. The influence of improvement in one mental function upon the efficiency of other functions.(i). *Psychological review*, 8(3):247, 1901.

[38] Yifan Wu, George Tucker, and Ofir Nachum. Behavior regularized offline reinforcement learning. *arXiv preprint arXiv:1911.11361*, 2019.

[39] Brian D Ziebart. Modeling purposeful adaptive behavior with the principle of maximum causal entropy. 2010.

A Action-prior Regularized Soft Actor-Critic

The original derivation of the SAC algorithm assumes a uniform prior over actions. We extend the formulation to the case with a non-uniform action prior $p(a|\cdot)$, where the dot indicates that the prior can be non-conditional or conditioned on e.g., the current state or the previous action. Our derivation closely follows Haarnoja et al. [10] and Levine [18] with the key difference that we replace the entropy maximization in the reward function with a term that penalizes divergence from the action prior. We derive the formulation for single-step action priors below, and the extension to *skill priors* is straightforward by replacing actions a_t with skill embeddings z_t.

We adopt the probabilistic graphical model (PGM) described in [18], which includes *optimality variables* $\mathcal{O}_{1:T}$, whose distribution is defined as $p(\mathcal{O}_t|s_t, a_t) = \exp\big(r(s_t, a_t)\big)$ where $r(s_t, a_t)$ is the reward. We treat $\mathcal{O}_{1:T} = 1$ as evidence in our PGM and obtain the following conditional trajectory distribution:

$$\begin{aligned} p(\tau|\mathcal{O}_{1:T}) &= p(s_1)\prod_{t=1}^{T} p(\mathcal{O}_t|s_t, a_t)p(s_{t+1}|s_t, a_t)p(a_t|\cdot) \\ &= \left[p(s_1)\prod_{t=1}^{T} p(s_{t+1}|s_t, a_t)p(a_t|\cdot)\right] \cdot \exp\sum_{t=1}^{T} r(s_t, a_t) \end{aligned}$$

Crucially, in contrast to [18] we did not omit the action prior $p(a_t|\cdot)$ since we assume it to be generally not uniform.

Our goal is to derive an objective for learning a policy that induces such a trajectory distribution. Following [18] we will cast this problem within the framework of structured variational inference and derive an expression for an evidence lower bound (ELBO).

We define a variational distribution $q(a_t|s_t)$ that represents our policy. It induces a trajectory distribution $q(\tau) = p(s_1)\prod_{t=1}^{T} p(s_{t+1}|s_t, a_t)q(a_t|s_t)$. We can derive the ELBO as:

$$\begin{aligned} \log p(\mathcal{O}_{1:T}) &\geq -D_{\text{KL}}\big[q(\tau) \,||\, p(\tau|\mathcal{O}_{1:T})\big] \\ &\geq \mathbb{E}_{\tau\sim q(\tau)}\Bigg[\log p(s_1) + \sum_{t=1}^{T}\big[\log p(s_{t+1}|s_t, a_t)\log p(a_t|\cdot)\big] + \sum_{t=1}^{T} r(s_t, a_t) \\ &\qquad - \log p(s_1) - \sum_{t=1}^{T}\big[\log p(s_{t+1}|s_t, a_t)\log q(a_t|s_t)\big]\Bigg] \\ &\geq \mathbb{E}_{\tau\sim q(\tau)}\left[\sum_{t=1}^{T} r(s_t, a_t) + \log p(a_t|\cdot) - \log q(a_t|s_t)\right] \\ &\geq \mathbb{E}_{\tau\sim q(\tau)}\left[\sum_{t=1}^{T} r(s_t, a_t) - D_{\text{KL}}\big[q(a_t|s_t) \,||\, p(a_t|\cdot)\big]\right] \end{aligned}$$

Note that in the case of a uniform action prior the KL divergence is equivalent to the negative entropy $-\mathcal{H}(q(a_t|s_t))$. Substituting the KL divergence with the entropy recovers the ELBO derived in [18].

To maximize this ELBO with respect to the policy $q(a_t|s_t)$, [18] propose to use an inference procedure based on a message passing algorithm. Following this derivation for the "messages" $V(s_t)$ and $Q(s_t, a_t)$ (Levine [18], section 4.2), but substituting policy entropy $-\log q(a_t|s_t)$ with prior divergence $D_{\text{KL}}\big[q(a_t|s_t) \,||\, p(a_t|\cdot)\big]$, the **modified Bellman backup operator** can be derived as:

$$\begin{aligned} \mathcal{T}^{\pi}Q(s_t, a_t) &= r(s_t, a_t) + \gamma\mathbb{E}_{s_{t+1}\sim p}\big[V(s_{t+1})\big] \\ \text{where } V(s_t) &= \mathbb{E}_{a_t\sim\pi}\big[Q(s_t, a_t) - D_{\text{KL}}\big[\pi(a_t|s_t) \,||\, p(a_t|\cdot)\big]\big] \end{aligned}$$

To show convergence of this operator to the optimal Q function we follow the proof of [10] in appendix B1 and introduce a divergence-augmented reward:

$$r_{\pi}(s_t, a_t) = r(s_t, a_t) - \mathbb{E}_{s_{t+1}\sim p}\Big[D_{\text{KL}}\big[\pi(a_{t+1}|s_{t+1}) \,||\, p(a_{t+1}|\cdot)\big]\Big].$$

Algorithm 1 Skill-prior regularized Soft Actor-Critic

Inputs: high-level policy $\pi_\theta(z_t|s_t)$, critic $Q_\phi(s_t, z_t)$, target network $Q_{\bar{\phi}}(s_t, z_t)$, H-step reward function $\tilde{r}(s_t, z_t)$, discount γ, target divergence δ, learning rates $\lambda_\pi, \lambda_Q, \lambda_\alpha$, target update τ.
Initialize empty replay buffer: $\mathcal{D} \leftarrow \emptyset$
for each iteration **do**
  for each environment step **do**
    $z_t \sim \pi(z_t|s_t)$ ▷ sample skill from policy
    $s_{t'} \sim p(s_{t+H}|s_t, z_t)$ ▷ execute skill in environment
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{s_t, z_t, \tilde{r}(s_t, z_t), s_{t'}\}$ ▷ store transition in replay buffer
  for each gradient step **do**
    $\bar{Q} = \tilde{r}(s_t, z_t) + \gamma \big[Q_{\bar{\phi}}(s_{t'}, \pi_\theta(z_{t'}|s_{t'})) - \alpha D_{\text{KL}}\big(\pi_\theta(z_{t'}|s_{t'}), p_{\boldsymbol{a}}(z_{t'}|s_{t'})\big) \big]$ ▷ compute Q-target
    $\theta \leftarrow \theta - \lambda_\pi \nabla_\theta \big[Q_\phi(s_t, \pi_\theta(z_t|s_t)) - \alpha D_{\text{KL}}(\pi_\theta(z_t|s_t), p_{\boldsymbol{a}}(z_t|s_t)) \big]$ ▷ update policy weights
    $\phi \leftarrow \phi - \lambda_Q \nabla_\phi \big[\frac{1}{2} \big(Q_\phi(s_t, z_t) - \bar{Q} \big)^2 \big]$ ▷ update critic weights
    $\alpha \leftarrow \alpha - \lambda_\alpha \nabla_\alpha \big[\alpha \cdot (D_{\text{KL}}(\pi_\theta(z_t|s_t), p_{\boldsymbol{a}}(z_t|s_t)) - \delta) \big]$ ▷ update alpha
    $\bar{\phi} \leftarrow \tau\phi + (1-\tau)\bar{\phi}$ ▷ update target network weights
return trained policy $\pi_\theta(z_t|s_t)$

Then we can recover the original Bellman update:

$$Q(s_t, a_t) \leftarrow r_\pi(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim p, a_{t+1} \sim \pi} \big[Q(s_{t+1}, a_{t+1}) \big],$$

for which the known convergence guarantees hold [32].

The modifications to the messages $Q(s_t, a_t)$ and $V(s_t)$ directly lead to the following **modified policy improvement operator**:

$$\arg\min_\theta \mathbb{E}_{s_t \sim \mathcal{D}, a_t \sim \pi} \Big[D_{\text{KL}} \big[\pi(a_t|s_t) \;||\; p(a_t|\cdot) \big] - Q(s_t, a_t) \Big]$$

Finally, the practical implementations of SAC introduce a temperature parameter α that trades off between the reward and the entropy term in the original formulation and the reward and divergence term in our formulation. Haarnoja et al. [11] propose an algorithm to automatically adjust α by formulating policy learning as a constrained optimization problem. In our formulation we derive a similar update mechanism for α. We start by formulating the following constrained optimization problem:

$$\max_{x_{1:T}} \mathbb{E}_{p_\pi} \Big[\sum_{t=1}^{T} r(s_t, a_t) \Big] \quad \text{s.t. } D_{\text{KL}} \big[\pi(a_t|s_t) \;||\; p(a_t|\cdot) \big] \leq \delta \quad \forall t$$

Here δ is a target divergence between policy and action prior similar to the target entropy $\bar{H}$ is the original SAC formulation. We can formulate the dual problem by introducing the temperature α:

$$\min_{\alpha > 0} \max_\pi \sum_{t=1}^{T} r(s_t, a_t) + \alpha \big(\delta - D_{\text{KL}} \big[\pi(a_t|s_t) \;||\; p(a_t|\cdot) \big] \big)$$

This leads to the **modified update objective for** α:

$$\arg\min_{\alpha > 0} \mathbb{E}_{a_t \sim \pi} \big[\alpha\delta - \alpha D_{\text{KL}} \big[\pi(a_t|s_t) \;||\; p(a_t|\cdot) \big] \big]$$

We combine the modified objectives for Q-value function, policy and temperature α in the skill-prior regularized SAC algorithm, summarized in algorithm 1.

B Environment and Comparison Details

We evaluate our approach on one simulated navigation task and two simulated robotic manipulation tasks (see Fig. 4). For each environment, we collect a large and diverse dataset of agent experience that allows to extract a large number of skills. To test our method's ability to transfer to *unseen* downstream tasks, we vary task and environment setup between training data collection and downstream task.

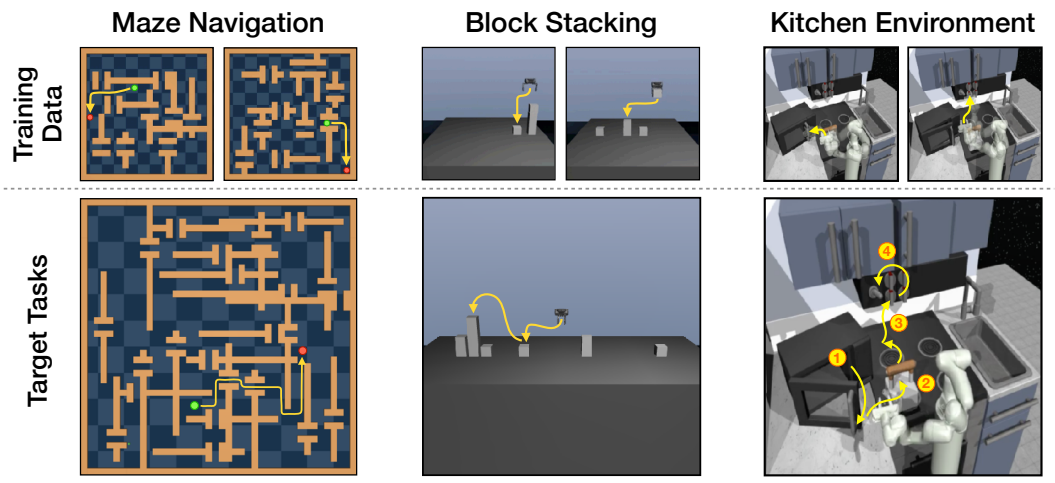

Figure 4: For each environment we collect a diverse dataset from a wide range of training tasks (examples on **top**) and test skill transfer to more complex target tasks (**bottom**), in which the agent needs to: navigate a maze (**left**), stack as many blocks as possible (**middle**) and manipulate a kitchen setup to reach a target configuration (**right**). All tasks require the execution of complex, long-horizon behaviors and need to be learned from sparse rewards.

Maze Navigation. A simulated maze navigation environment based on the D4RL maze environment [6]. The task is to navigate a point mass agent through a maze between fixed start and goal locations. We use a planner-based policy to collect $85\,000$ goal-reaching trajectories in randomly generated, small maze layouts and test generalization to a goal-reaching task in a randomly generated, larger maze. The state is represented as a RGB top-down view centered around the agent. For downstream learning the agent only receives a sparse reward when in close vicinity to the goal. The agent can transfer skills like traversing hallways or passing through narrow doors, but needs to learn to navigate a new maze layout for solving the downstream task.

Block Stacking. The goal of the agent is to stack as many blocks as possible in an environment with eleven blocks. We collect $37\,000$ training sequences with a noisy, scripted policy that randomly stacks blocks on top of each other in a smaller environment with only five blocks. The state is represented as a RGB front view centered around the agent and it receives binary rewards for picking up and stacking blocks. The agent can transfer skills like picking up, carrying and stacking blocks, but needs to perform a larger number of consecutive stacks than seen in the training data on a new environment with more blocks.

Kitchen Environment. A simulated kitchen environment based on Gupta et al. [9]. We use the training data provided in the D4RL benchmark [6], which consists of 400 teleoperated sequences in which the 7-DoF robot arm manipulates different parts of the environment (e.g., open microwave, switch on stove, slide cabinet door). During downstream learning the agent needs to execute an unseen sequence of multiple subtasks. It receives a sparse, binary reward for each successfully completed manipulation. The agent can transfer a rich set of manipulation skills, but needs to recombine them in new ways to solve the downstream task.

For further details on environment setup, data collection and training, see appendix, sections C and D.

We compare the downstream task performance of our approach to several flat and hierarchical baselines that test the importance of learned skill embeddings and skill prior:

- **Flat Model-Free RL (SAC).** Trains an agent *from scratch* with Soft Actor-Critic (SAC, [10]). This comparison tests the benefit of leveraging prior experience.
- **Behavioral Cloning w/ finetuning (BC + SAC).** Trains a supervised behavioral cloning (BC) policy from the offline data and finetunes it on the downstream task using SAC.

- **Flat Behavior Prior (Flat Prior).** Learns a single-step action prior on the primitive action space and uses it to regularize downstream learning as described in section 3.2, similar to [31]. This comparison tests the importance of temporal abstraction through learned skills.
- **Hierarchical Skill-Space Policy (SSP).** Trains a high-level policy on the skill-embedding space of the model described in section 3.1 but without skill prior, representative of [22, 16, 29]. This comparison tests the importance of the learned skill prior for downstream task learning.

C Implementation Details

C.1 Model Architecture and Training Objective

We instantiate the skill embedding model described in section 3.1 with deep neural networks. The skill encoder is implemented as a one-layer LSTM with 128 hidden units. After processing the full input action sequence, it outputs the parameters (μ_z, σ_z) of the Gaussian posterior distribution in the 10-dimensional skill embedding space $\mathcal{Z}$. The skill decoder mirrors the encoder's architecture and is unrolled for H steps to produce the H reconstructed actions. The sampled skill embedding z is passed as input in every step.

The skill prior is implemented as a 6-layer fully-connected network with 128 hidden units per layer. It parametrizes the Gaussian skill prior distribution $\mathcal{N}(\mu_p, \sigma_p)$. For image-based state inputs in maze and block stacking environment, we first pass the state through a convolutional encoder network with three layers, a kernel size of three and $(8, 16, 32)$ channels respectively. The resulting feature map is flattened to form the input to the skill prior network.

We use leaky-ReLU activations and batch normalization throughout our architecture. We optimize our model using the RAdam optimizer with parameters with $\beta_1 = 0.9$ and $\beta_2 = 0.999$, batch size 16 and learning rate $1\text{e}{-3}$. Training on a single high-end NVIDIA GPU takes approximately 8 hours. Assuming a unit-variance Gaussian output distribution our full training objective is:

$$\mathcal{L} = \sum_{i=1}^{H} \underbrace{(a_i - \hat{a}_i)^2}_{\text{reconstruction}} - \beta \underbrace{D_{\text{KL}}\big(\mathcal{N}(\mu_z, \sigma_z) || \mathcal{N}(0, I)\big)}_{\text{regularization}} + \underbrace{D_{\text{KL}}\big(\mathcal{N}(\lfloor \mu_z \rfloor, \lfloor \sigma_z \rfloor) || \mathcal{N}(\mu_p, \sigma_p)\big)}_{\text{prior training}}. \quad (1)$$

Here $\lfloor \cdot \rfloor$ indicates that gradients flowing through these variables are stopped. For Gaussian distributions the KL divergence can be analytically computed. For non-Gaussian prior parametrizations (e.g. with Gaussian mixture model or normalizing flow priors) we found that sampling-based estimates also suffice to achieve reliable, albeit slightly slower convergence. We tune the weighting parameter β separately for each environment and use $\beta = 1\text{e}{-2}$ for maze and block stacking and $\beta = 5\text{e}{-4}$ for the kitchen environment.

C.2 Reinforcement Learning Setup

The architecture of policy and critic mirror the one of the skill prior network. The policy outputs the parameters of a Gaussian action distribution while the critic outputs a single Q-value estimate. Empirically, we found it important to initialize the weights of the policy with the pre-trained skill prior weights in addition to regularizing towards the prior.

We use the hyperparameters of the standard SAC implementation [10] with batch size 256, replay buffer capacity of $1\text{e}6$ and discount factor $\gamma = 0.99$. We collect $5\text{e}3$ warmup rollout steps to initialize the replay buffer before training. We use Adam optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.999$ and learning rate $3\text{e}{-4}$ for updating policy, critic and temperature α. Analogous to SAC, we train two separate critic networks and compute the Q-value as the minimum over both estimates to stabilize training. The corresponding target networks get updated at a rate of $\tau = 5\text{e}{-3}$. The policies' action range is limited in the range $[-2 \dots 2]$ by a $\tanh$ "squashing function" (see Haarnoja et al. [10], appendix C).

We tune the target divergence δ separately for each environment and use $\delta = 1$ for the maze navigation task and $\delta = 5$ for both robot manipulation tasks.

D Environments and Data Collection

Maze Navigation. The maze navigation environment is based on the maze environment in the D4RL benchmark [6]. Instead of using a single, fixed layout, we generate random layouts for training data collection by placing walls with doorways in randomly sampled positions. For each collected training sequence we sample a new maze layout and randomly sample start and goal position for the agent. Following Fu et al. [6], we collect goal-reaching examples through a combination of high-level planner with access to a map of the maze and a low-level controller that follows the plan.

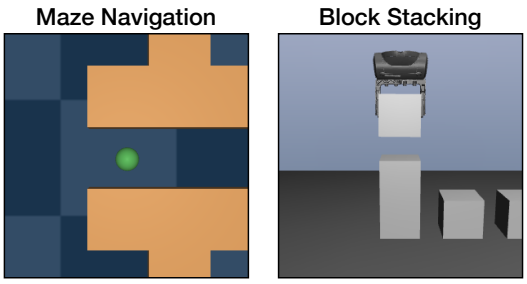

Figure 5: Image-based state representation for maze (**left**) and block stacking (**right**) environment.

For the downstream task we randomly sample a maze that is four times larger than the training data layouts. We keep maze layout, as well as start and goal location for the agent fixed throughout downstream learning. The policy outputs (x,y)-velocities for the agent. The state is represented as a local top-down view around the agent (see Fig. 5). To represent the agent's velocity, we stack two consecutive 32×32px observations as input to the policy. The agent receives a per-timestep binary reward when the distance to the goal is below a threshold.

Block Stacking. The block stacking environment is simulated using the Mujoco physics engine. For data collection, we initialize the five blocks in the environment to be randomly stacked on top of each other or placed at random locations in between. We use a hand-coded data collection policy to generate trajectories with up to three consecutive stacking manipulations. The location of blocks and the movement of the agent are limited to a 2D plane and a barrier prevents the agent from leaving the table. To increase the support of the collected trajectories we add noise to the hard-coded policy by placing pseudo-random subgoals in between and within stacking sequences.

The downstream task of the agent is to stack as many blocks as possible in a larger version of the environment with 11 blocks. The environment state is represented through a front view of the agent (see Fig. 5). The policies' input is a stack of two 32×32px images and it outputs (x,z)-displacements for the robot as well as a continuous action in range $[0 \dots 1]$ that represents the opening degree of the gripper. The agent receives per-timestep binary rewards for lifting a block from the ground and moving it on top of another block. It further receives a reward proportional to the height of the highest stacked tower.

Kitchen environment. We use the kitchen environment from the D4RL benchmark [6] which was originally published by Gupta et al. [9]. For training we use the data provided in D4RL (dataset version "mixed"). It consists of trajectories collected via human tele-operation that each perform four consecutive manipulations of objects in the environment. There are seven manipulatable objects in the environment. The downstream task of the agent consists of performing an *unseen* sequence of four manipulations - while the individual manipulations have been observed in the training data, the agent needs to learn to recombine these skills in a new way to solve the task. The state is a 30-dimensional vector representing the agent's joint velocities as well as poses of the manipulatable objects. The agent outputs 7-dimensional joint velocities for robot control as well as a 2-dimensional continuous gripper opening/closing action. It receives a one-time reward whenever fulfilling one of the subtasks.

E Ablation Studies

We analyze the influence of skill horizon H and dimensionality of the learned skill space $|\mathcal{Z}|$ on downstream performance in Fig. 8. We see that too short skill horizons do not afford sufficient temporal abstraction. Conversely, too long horizons make the skill exploration problem harder, since a larger number of possible skills gets embedded in the skill space. Therefore, the policy converges slower.

We find that the dimensionality of the learned skill embedding space needs to be large enough to represent a sufficient diversity of skills. Beyond that, $|\mathcal{Z}|$ does not have a major influence on the downstream performance. We attribute this to the usage of the learned skill prior: even though the

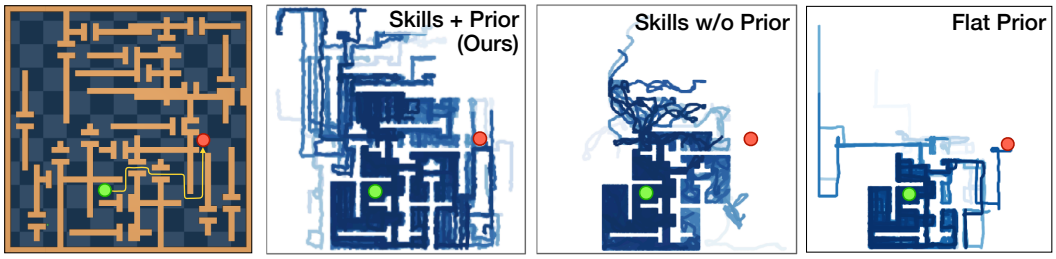

Figure 6: Exploration behavior of our method vs. alternative transfer approaches on the downstream maze task. Through learned skill embeddings and skill priors our method can explore the environment more widely. We visualize positions of the agent during 1M steps of exploration rollouts in blue and mark episode start and goal positions in green and red respectively.

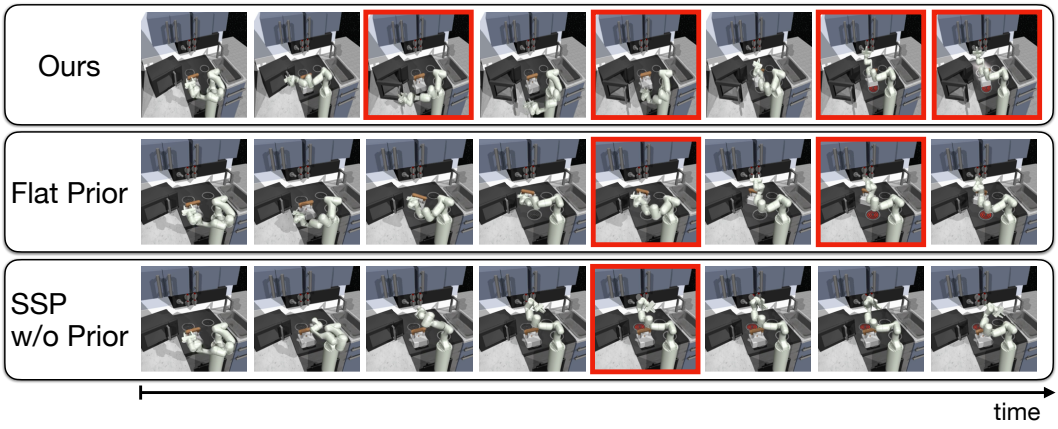

Figure 7: Comparison of policy execution traces on the kitchen environment. Following Fu et al. [6], the agent's task is to (1) open the microwave, (2) move the kettle backwards, (3) turn on the burner and (4) switch on the light. Red frames mark the completion of subtasks. Our skill-prior guided agent (**top**) is able to complete all four subtasks. In contrast, the agent using a flat single-action prior (**middle**) only learns to solve two subtasks, but lacks temporal abstraction and hence fails to solve the complete long-horizon task. The skill-space policy without prior guidance (**bottom**) cannot efficiently explore the skill space and gets stuck in a local optimum in which it solves only a single subtask. Best viewed electronically and zoomed in.

nominal dimensionality of the high-level policies' action space increases, its effective dimensionality remains unchanged since the skill prior focuses exploration on the relevant parts of the skill space.

F Reuse of Learned Skill Priors

Our approach has two separate stages: (1) learning of skill embedding and skill prior from offline data and (2) prior-regularized downstream RL. Since the learning of the skill prior is *independent* of the downstream task, we can reuse the same skill prior for guiding learning on multiple downstream tasks. To test this, we learn a single skill prior on the maze environment depicted in Fig. 4 (left) and use it to train multiple downstream task agents that reach different goals.

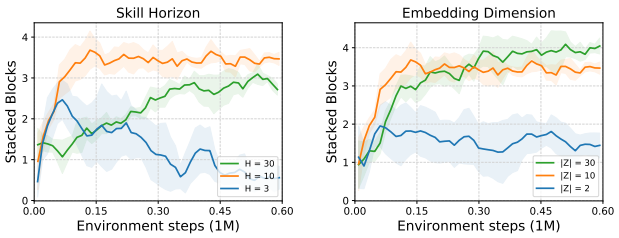

Figure 8: Ablation analysis of skill horizon and skill space dimensionality on block stacking task. See text for details.

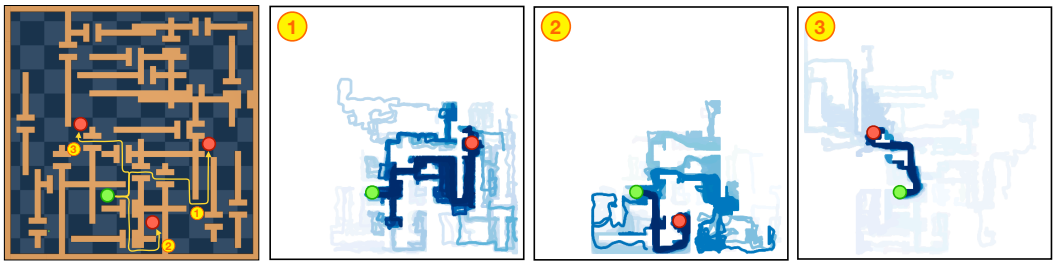

Figure 9: Reuse of one learned skill prior for multiple downstream tasks. We train a single skill embedding and skill prior model and then use it to guide downstream RL for multiple tasks. **Left**: We test prior reuse on three different maze navigation tasks in the form of different goals that need to be reached. **(1)-(3)**: Agent rollouts during training; the darker the rollout paths, the later during training they were collected. The same prior enables efficient exploration for all three tasks, but allows for convergence to task-specific policies that reach each of the goals upon convergence.

In Fig. 9 we show a visualization of the training rollouts in a top-down view, similar to the visualization in Fig. 6; darker trajectories are more recent. We can see that the same prior is able to guide downstream agents to efficiently learn to reach diverse goals. All agents achieve $\sim 100\,\%$ success rate upon convergence. Intuitively, the prior captures the knowledge that it is more meaningful to e.g. cross doorways instead of crashing into walls, which helps exploration in the maze independent of the goal position.