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; for example information about the current state can hint which skills are promising to explore. In this work, 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 use skill priors to guide downstream learning. We validate our approach, SPiRL (Skill-Prior RL), on complex navigation and robotic manipulation tasks and show that learned skill priors are essential for effective skill transfer from rich datasets. Videos and code are available at `clvrai.com/spirl`.

Keywords: Reinforcement Learning, Skill Learning, Transfer Learning

1   Introduction

Intelligent agents are able to utilize a large pool of prior experience to efficiently learn how to solve new tasks [1]. In contrast, reinforcement learning (RL) agents typically learn each new task *from scratch*, without leveraging prior experience. Consequently, agents need to collect a large amount of experience while learning the target task, which is expensive, especially in the real world. On the other hand, there is an abundance of collected agent experience available in domains like autonomous driving [2], indoor navigation [3], or robotic manipulation [4, 5]. With the widespread deployment of robots on streets or in warehouses the available amount of data will further increase in the future. However, the majority of this data is unstructured, without clear task or reward definitions, making it difficult to use for learning new tasks. 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. Skills can be learned from data without any task or reward information and can be transferred to new tasks and even new environment configurations. Prior work has learned skill libraries from data collected by humans [6, 7, 8, 9, 10] or by agents autonomously exploring the world [11, 12]. 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. Thus, the large skill libraries extracted from rich datasets can, somewhat paradoxically, lead to worse learning efficiency on the downstream task, since the agent needs to collect large amounts of experience to perform the necessary exploration in the space of skills [13].

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 kettle, it is more promising
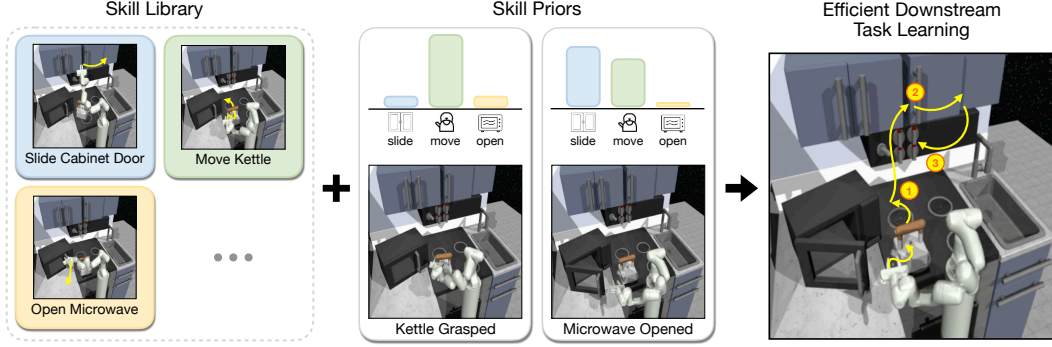
Figure 1: Intelligent agents can use a large library of acquired skills when learning new tasks. Instead of exploring skills uniformly, they can leverage *priors over skills* as guidance, based e.g., on the current environment state. Such priors capture which skills are promising to explore, like moving 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.

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 skill prior into maximum-entropy RL algorithms for efficient learning of downstream tasks. To validate the effectiveness of our approach, SPiRL (Skill-Prior RL), we apply it to 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 downstream tasks and that learned skill priors are essential to efficiently utilize rich experience datasets.

In summary, our contributions 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 downstream task learning, and (3) we show that learned skill priors accelerate learning of new tasks across three simulated navigation and robot manipulation tasks.

2 Related Work

The goal of our work is to leverage prior experience for accelerated learning of downstream tasks. **Meta-learning** approaches [14, 15] similarly aim to extract useful priors from previous experience to improve the learning efficiency for unseen tasks. However, they require a defined set of training tasks and online data collection during pre-training and therefore cannot leverage large offline datasets. In contrast, our model learns skills fully offline from unstructured data.

Approaches that operate on such offline data are able to leverage large existing datasets [4, 5] and can be applied to domains where data collection is particularly costly or safety critical [16]. A number of works have recently explored the **offline reinforcement learning** setting [16, 17, 18, 19, 20], in which a task needs to be learned purely from logged agent experience without any environment interactions. It has also been shown how offline RL can be used to accelerate online RL [21]. However, these approaches require the experience to be annotated with rewards for the downstream task, which are challenging to provide for large, real-world datasets, especially when the experience is collected across a wide range of tasks. Our approach based on skill extraction, on the other hand, does not require any reward annotation on the offline experience data and, once extracted, skills can be reused for learning a wide range of downstream tasks.

More generally, the problem of inter-task transfer has been studied for a long time in the RL community [22]. The idea of **transferring skills between tasks** dates back at least to the SKILLS [23] and *PolicyBlocks* [24] algorithms. Learned skills can be represented as sub-policies in the form of options [25, 26], as subgoal setter and reacher functions [27, 28] or as discrete primitive libraries [6, 29]. Recently, a number of works have explored the embedding of skills into a continuous *skill space*

via stochastic latent variable models [11, 7, 30, 8, 9, 31, 10]. 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 [13]. 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 [17, 18, 20] 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 [32]. 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 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. In this work, we propose to use learned skill priors to guide exploration in skill space and allow for efficient skill transfer from large datasets. We decompose the problem of prior-guided skill transfer into two sub-problems: (1) the extraction of skill embedding and skill prior from offline data, and (2) the prior-guided learning of downstream tasks with a hierarchical policy.

3.1 Problem Formulation

We assume access to a dataset $\mathcal{D}$ of pre-recorded agent experience in the form of state-action trajectories $\tau_i = \{(s_0, a_0), \dots, (s_{T_i}, a_{T_i})\}$. This data can be collected using previously trained agents across a diverse set of tasks [33, 34], through agents autonomously exploring their environment [11, 12], via human teleoperation [35, 27, 36, 10] 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. In contrast to imitation learning problems we do not assume that the training data contains complete solutions for the downstream task. Hence, we focus on transferring skills to new problems.

The downstream learning problem is formulated as a Markov decision process (MDP) defined by a tuple $\{\mathcal{S}, \mathcal{A}, \mathcal{T}, R, \rho, \gamma\}$ of states, actions, transition probability, reward, initial state distribution, and discount factor. We aim to learn a policy $\pi_\theta(a|s)$ with parameters θ that maximizes the discounted sum of rewards $J(\theta) = \mathbb{E}_\pi \left[\sum_{t=0}^{T-1} \gamma^t r_t \right]$ where T is the episode horizon.

3.2 Learning Continuous Skill Embedding and Skill Prior

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 [7, 8, 27, 31, 28, 37]. Other work has proposed to learn semantic skills of flexible length [30, 9, 38] and our model can be extended to include similar approaches, but we leave this for future work.

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):

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

Here, β is a parameter that is commonly used to tune the weight of the regularization term [39]. We optimize this objective using amortized variational inference with an inference network $q(z|\boldsymbol{a}_i)$ [40, 41]. To learn a rich skill embedding space we implement skill encoder $q(z|\boldsymbol{a}_i)$ and decoder $p(\boldsymbol{a}_i|z)$ as deep neural networks that output the parameters of the Gaussian posterior and output distributions. The prior $p(z)$ is set to be unit Gaussian $\mathcal{N}(0, I)$. Once trained, we can sample skills from our model by sampling latent variables $z \sim \mathcal{N}(0, I)$ and passing them through the decoder $p(\boldsymbol{a}_i|z)$. In Section 3.3 we show how to use this generative model of skills for learning hierarchical RL policies.
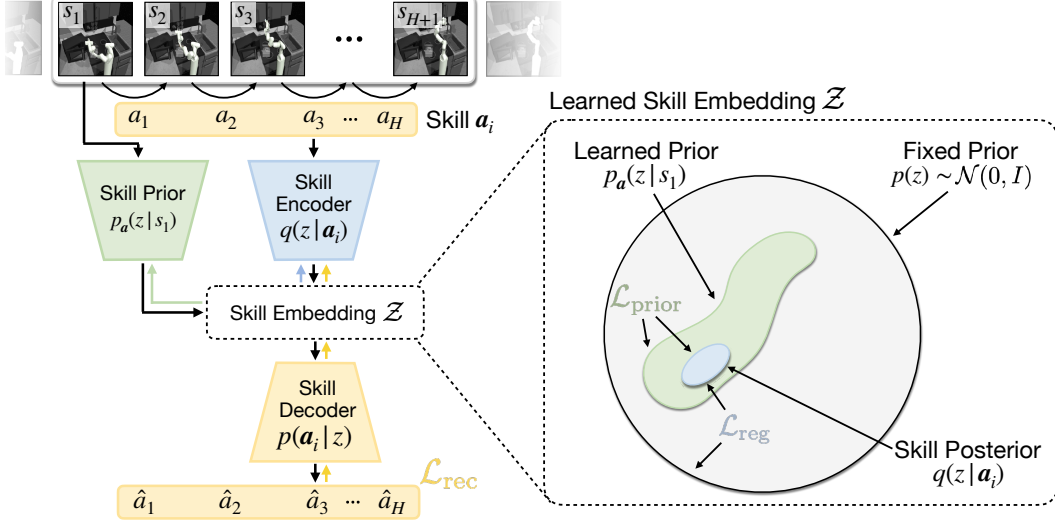
3

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.

To better guide downstream learning, we 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 conditioning of this skill prior can be adjusted to the environment and task at hand, but should be informative about the set of skills that are meaningful to explore in a given situation. Possible choices include the embedding of the last executed skill z_{t-1} or the current environment state s_t. 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, 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: $\mathbb{E}_{(s,\boldsymbol{a}_i)\sim\mathcal{D}}D_{\mathrm{KL}}\big(q(z|\boldsymbol{a}_i), p_{\boldsymbol{a}}(z|s_t)\big)$. Using the reverse KL divergence $D_{\mathrm{KL}}(q, p)$ instead of $D_{\mathrm{KL}}(p, q)$ ensures that the learned prior is mode-covering [42], i.e., represents *all* observed skills in the current situation. Instead of training the skill prior after training the skill embedding model, we can jointly optimize both models and ensure stable convergence by stopping gradients from the skill prior objective into the skill encoder. We experimented with different parameterizations for the skill prior distribution, in particular multi-modal distributions, such as Gaussian mixture models and normalizing flows [43, 44], but found simple Gaussian skill priors to work equally well in our experiments. For further implementation details, see appendix, Section B.

3.3 Skill Prior Regularized Reinforcement Learning

To use the learned skill embedding for downstream task learning, we employ a hierarchical policy learning scheme by using the skill embedding space as action space of a high-level policy. Concretely, instead of learning a policy over actions $a \in \mathcal{A}$ we learn a policy $\pi_\theta(z|s_t)$ that outputs skill embeddings, which we decode into action sequences using the learned skill decoder $\{a_t^i, \ldots, a_{t+H-1}^i\} \sim p(\boldsymbol{a}_i|z)$[1]. We execute these actions for H steps before sampling the next skill from the high-level policy. This hierarchical structure allows for temporal abstraction, which facilitates long-horizon task learning [25].

---

[1]We also experimented with models that directly condition the decoder on the current state, but found downstream RL to be less stable (see appendix, Section D)

Algorithm 1 SPiRL: Skill-Prior RL

1: **Inputs:** 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 rate τ.
2: Initialize replay buffer $\mathcal{D}$, high-level policy $\pi_\theta(z_t|s_t)$, critic $Q_\phi(s_t, z_t)$, target network $Q_{\bar{\phi}}(s_t, z_t)$
3: **for** each iteration **do**
4:     **for** every H environment steps **do**
5:         $z_t \sim \pi(z_t|s_t)$ ▷ sample skill from policy
6:         $s_{t'} \sim p(s_{t+H}|s_t, z_t)$ ▷ execute skill in environment
7:         $\mathcal{D} \leftarrow \mathcal{D} \cup \{s_t, z_t, \tilde{r}(s_t, z_t), s_{t'}\}$ ▷ store transition in replay buffer
8:     **for** each gradient step **do**
9:         $\bar{Q} = \tilde{r}(s_t, z_t) + \gamma \big[Q_{\bar{\phi}}(s_{t'}, \pi_\theta(z_{t'}|s_{t'})) - \alpha D_{\mathrm{KL}}\big(\pi_\theta(z_{t'}|s_{t'}), p_{\boldsymbol{a}}(z_{t'}|s_{t'})\big) \big]$ ▷ compute Q-target
10:         $\theta \leftarrow \theta - \lambda_\pi \nabla_\theta \big[Q_\phi(s_t, \pi_\theta(z_t|s_t)) - \alpha D_{\mathrm{KL}}(\pi_\theta(z_t|s_t), p_{\boldsymbol{a}}(z_t|s_t)) \big]$ ▷ update policy weights
11:         $\phi \leftarrow \phi - \lambda_Q \nabla_\phi \big[\frac{1}{2} \big(Q_\phi(s_t, z_t) - \bar{Q} \big)^2 \big]$ ▷ update critic weights
12:         $\alpha \leftarrow \alpha - \lambda_\alpha \nabla_\alpha \big[\alpha \cdot (D_{\mathrm{KL}}(\pi_\theta(z_t|s_t), p_{\boldsymbol{a}}(z_t|s_t)) - \delta) \big]$ ▷ update alpha
13:         $\bar{\phi} \leftarrow \tau\phi + (1 - \tau)\bar{\phi}$ ▷ update target network weights
14: **return** trained policy $\pi_\theta(z_t|s_t)$

We can cast the problem of learning the high-level policy into a standard MDP by replacing the action space $\mathcal{A}$ with the skill space $\mathcal{Z}$, single-step rewards with H-step rewards $\tilde{r} = \sum_{t=1}^{H} r_t$, and single-step transitions with H-step transitions $s_{t+H} \sim p(s_{t+H}|s_t, z_t)$. We can then use conventional model-free RL approaches to maximize the return of the high-level policy $\pi_\theta(z|s_t)$.

This naive approach struggles when training a policy on a very rich skill space $\mathcal{Z}$ that encodes many different skills. While the nominal dimensionality of the skill space might be small, its continuous nature allows the model to embed an arbitrary number of different behaviors. Therefore, the *effective* dimensionality of the high-level policies' action space scales with the number of embedded skills. When using a large offline dataset $\mathcal{D}$ with diverse behaviors, the number of embedded skills can grow rapidly, leading to a challenging exploration problem when training the high-level policy. For more efficient exploration, we propose to use the learned skill prior to guide the high-level policy. We will next show how the skill prior can be naturally integrated into maximum-entropy RL algorithms.

Maximum entropy RL [45, 46] augments the training objective of the policy with a term that encourages maximization of the policy's entropy along with the return:

$$J(\theta) = \mathbb{E}_\pi \left[\sum_{t=1}^{T} \gamma^t r(s_t, a_t) + \alpha \mathcal{H}\big(\pi(a_t|s_t)\big) \right] \tag{2}$$

The added entropy term is equivalent to the negated KL divergence between the policy and a *uniform* action prior $U(a_t)$: $\mathcal{H}(\pi(a_t|s_t)) = -\mathbb{E}_\pi \big[\log \pi(a_t|s_t) \big] \propto -D_{\mathrm{KL}}(\pi(a_t|s_t), U(a_t))$ up to a constant. However, in our case we aim to regularize the policy towards a *non-uniform*, learned skill prior to guide exploration in skill space. We can therefore replace the entropy term with the negated KL divergence from the learned prior, leading to the following objective for the high-level policy:

$$J(\theta) = \mathbb{E}_\pi \left[\sum_{t=1}^{T} \tilde{r}(s_t, z_t) - \alpha D_{\mathrm{KL}}\big(\pi(z_t|s_t), p_{\boldsymbol{a}}(z_t|s_t)\big) \right] \tag{3}$$

We can modify the state-of-the-art maximum-entropy RL algorithms, such as Soft Actor-Critic (SAC [47, 48]) to optimize this objective. We summarize our SPiRL approach in Algorithm 1 with changes to SAC marked in red. For a detailed derivation of the update rules, see appendix, Section A. Analogous to Haarnoja et al. [48] we can devise an automatic tuning strategy for the regularization weight α by defining a *target divergence* parameter δ (see Algorithm 1, appendix A).

4 Experiments

Our experiments are designed to answer the following questions: (1) Can we leverage unstructured datasets to accelerate downstream task learning by transferring skills? (2) Can learned skill priors improve exploration during downstream task learning? (3) Are learned skill priors necessary to scale skill transfer to large datasets?
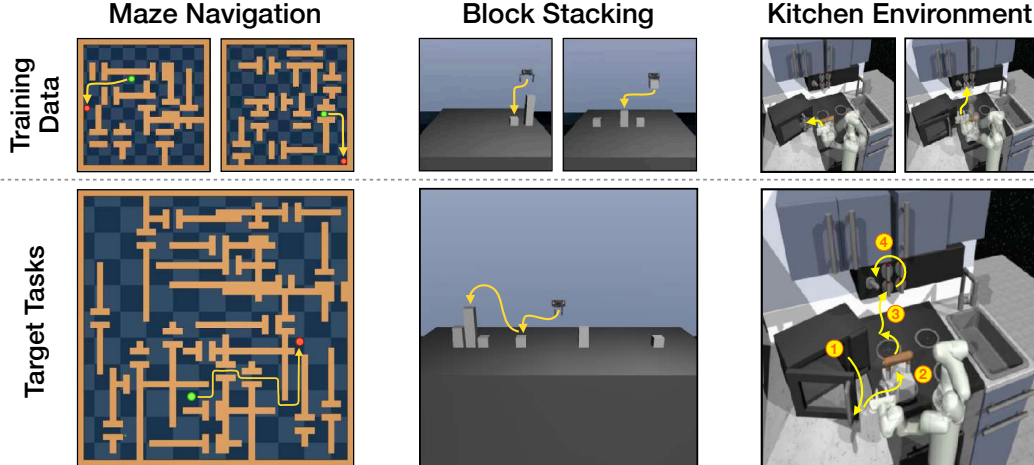
Figure 3: For each environment we collect a diverse dataset from a wide range of training tasks (examples on **top**) and test skill transfer to more complex target tasks (**bottom**), in which the agent needs to: navigate a maze (**left**), stack as many blocks as possible (**middle**) and manipulate 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.

4.1 Environments & Comparisons

We evaluate SPiRL on one simulated navigation task and two simulated robotic manipulation tasks (see Fig. 3). 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.

Maze Navigation. A simulated maze navigation environment based on the D4RL maze environment [33]. 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, such as 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. [27]. We use the training data provided in the D4RL benchmark [33], 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 B and C.
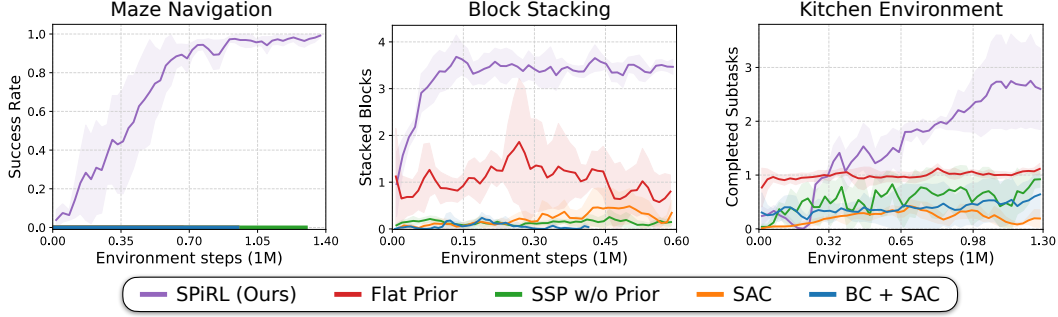
Figure 4: Downstream task learning curves for our method and all comparisons. Both, learned skill embedding and skill prior are essential for downstream task performance: single-action priors without temporal abstraction (**Flat Prior**) and learned skills without skill prior (**SSP w/o Prior**) fail to converge to good performance. Shaded areas represent standard deviation across three seeds.

We compare the downstream task performance of SPiRL 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, [47]). 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 3.3, similar to [32]. 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 3.2 but without skill prior, representative of [7, 30, 9]. This comparison tests the importance of the learned skill prior for downstream task learning.

4.2 Maze Navigation

We first evaluate SPiRL on the simulated maze navigation task. This task poses a hard exploration problem since the reward feedback is very sparse: following the D4RL benchmark [33] the agent receives a binary reward only when reaching the goal and therefore needs to explore large fractions of the maze without reward feedback. We hypothesize that learned skills and a prior that guides exploration are crucial for successful learning, particularly when external feedback is sparse.

In Fig. 4 (left) we show that only SPiRL is able to successfully learn a goal-reaching policy for the maze task; none of the baseline policies reaches the goal during training. To better understand this result, we compare the exploration behaviors of our approach and the baselines in Fig. 5: we collect rollouts by sampling from our skill prior and the single-step action prior and record the agent's position in the maze. To visualize the exploration behavior of skill-space policies without learned priors ("Skills w/o Prior") we sample skills uniformly from the skill space.

Fig. 5 shows that only SPiRL is able to explore large parts of the maze, since targeted sampling of skills from the prior guides the agent to navigate through doorways and traverse hallways. Random exploration in skill space, in contrast, does not lead to good exploration behavior since the agent often samples skills that lead to collisions. The comparison to single-step action priors ("Flat Prior") shows that temporal abstraction is beneficial for coherent exploration.

Finally, we show reuse of a single skill prior for a variety of downstream goals in appendix, Section H.

4.3 Robotic Manipulation

Next, we investigate the ability of SPiRL to scale to complex, robotic manipulation tasks in the block stacking problem and in the kitchen environment. For both environments we find that using learned skill embeddings together with the extracted skill prior is essential to solve the task (see Fig. 4, middle and right; appendix Fig. 8 for qualitative policy rollouts). In contrast, using non-hierarchical action

Figure 5: Exploration behavior of our method vs. alternative transfer approaches on the downstream maze task vs. random action sampling. 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.

priors ("Flat Prior") leads to performance similar to behavioral cloning of the training dataset, but fails to solve longer-horizon tasks. This shows the benefit of temporal abstraction through skills. 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 downstream learning. Instead, performance is comparable or worse than learning from scratch without skill transfer. This underlines the importance of learned skill priors for scaling skill transfer to large datasets. Similar to prior work [27], we find that a policy initialized through behavioral cloning is not amenable to efficient finetuning on complex, long-horizon tasks.

4.4 Ablation Studies

We analyze the influence of skill horizon H and dimensionality of the learned skill space $|\mathcal{Z}|$ on downstream performance in Fig. 6. 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.
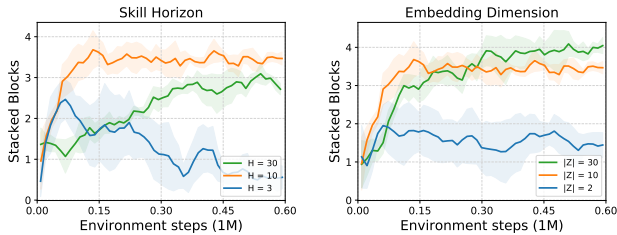

Figure 6: Ablation analysis of skill horizon and skill space dimensionality on block stacking task.

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

We further test the importance of prior initialization and regularization, as well as training priors from sub-optimal data in appendix, Sections E - G.

5   Conclusion

We present SPiRL, an approach for leveraging large, unstructured datasets to accelerate downstream learning of unseen tasks. We propose a deep latent variable model that jointly learns an embedding space of skills and a prior over these skills from offline data. We then extend maximum-entropy RL algorithms to incorporate both skill embedding and skill prior for efficient downstream learning. Finally, we evaluate SPiRL on challenging simulated navigation and robotic manipulation tasks and show that both, skill embedding and skill prior are essential for effective transfer from rich datasets.

Future work can combine learned skill priors with methods for extracting *semantic* skills of flexible length from unstructured data [9, 38]. Further, skill priors are important in safety-critical applications, like autonomous driving, where random exploration is dangerous. Skill priors learned e.g. from human demonstration, can guide exploration to skills that do not endanger the learner or other agents.

References

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

[2] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom. nuscenes: A multimodal dataset for autonomous driving. *preprint arXiv:1903.11027*, 2019.

[3] K. Mo, H. Li, Z. Lin, and J.-Y. Lee. The AdobeIndoorNav Dataset: Towards deep reinforcement learning based real-world indoor robot visual navigation. *preprint arXiv:1802.08824*, 2018.

[4] S. Dasari, F. Ebert, S. Tian, S. Nair, B. Bucher, K. Schmeckpeper, S. Singh, S. Levine, and C. Finn. Robonet: Large-scale multi-robot learning. *CoRL*, 2019.

[5] S. Cabi, S. G. Colmenarejo, A. Novikov, K. Konyushkova, S. Reed, R. Jeong, K. Zolna, Y. Aytar, D. Budden, M. Vecerik, O. Sushkov, D. Barker, J. Scholz, M. Denil, N. de Freitas, and Z. Wang. Scaling data-driven robotics with reward sketching and batch reinforcement learning. *RSS*, 2019.

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

[7] J. Merel, L. Hasenclever, A. Galashov, A. Ahuja, V. Pham, G. Wayne, Y. W. Teh, and N. Heess. Neural probabilistic motor primitives for humanoid control. *ICLR*, 2019.

[8] J. Merel, S. Tunyasuvunakool, A. Ahuja, Y. Tassa, L. Hasenclever, V. Pham, T. Erez, G. Wayne, and N. Heess. Catch & carry: Reusable neural controllers for vision-guided whole-body tasks. *ACM. Trans. Graph.*, 2020.

[9] T. Shankar, S. Tulsiani, L. Pinto, and A. Gupta. Discovering motor programs by recomposing demonstrations. In *International Conference on Learning Representations*, 2019.

[10] C. Lynch, M. Khansari, T. Xiao, V. Kumar, J. Tompson, S. Levine, and P. Sermanet. Learning latent plans from play. In *Conference on Robot Learning*, pages 1113–1132, 2020.

[11] K. Hausman, J. T. Springenberg, Z. Wang, N. Heess, and M. Riedmiller. Learning an embedding space for transferable robot skills. In *International Conference on Learning Representations*, 2018.

[12] A. Sharma, S. Gu, S. Levine, V. Kumar, and K. Hausman. Dynamics-aware unsupervised discovery of skills. *ICLR*, 2020.

[13] N. K. Jong, T. Hester, and P. Stone. The utility of temporal abstraction in reinforcement learning. In *AAMAS (1)*, pages 299–306. Citeseer, 2008.

[14] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *ICML*, 2017.

[15] K. Rakelly, A. Zhou, C. Finn, S. Levine, and D. Quillen. Efficient off-policy meta-reinforcement learning via probabilistic context variables. In *ICML*, 2019.

[16] S. Levine, A. Kumar, G. Tucker, and J. Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *arXiv preprint arXiv:2005.01643*, 2020.

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

[18] N. Jaques, A. Ghandeharioun, J. H. Shen, C. Ferguson, A. Lapedriza, N. Jones, S. Gu, and R. Picard. Way off-policy batch deep reinforcement learning of implicit human preferences in dialog. *arXiv preprint arXiv:1907.00456*, 2019.

[19] A. Kumar, J. Fu, M. Soh, G. Tucker, and S. Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. In *Advances in Neural Information Processing Systems*, pages 11784–11794, 2019.

[20] Y. Wu, G. Tucker, and O. Nachum. Behavior regularized offline reinforcement learning. *arXiv preprint arXiv:1911.11361*, 2019.

[21] A. Nair, M. Dalal, A. Gupta, and S. Levine. Accelerating online reinforcement learning with offline datasets. *arXiv preprint arXiv:2006.09359*, 2020.

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

[23] S. Thrun and A. Schwartz. Finding structure in reinforcement learning. In *NIPS*, 1995.

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

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

[26] P.-L. Bacon, J. Harb, and D. Precup. The option-critic architecture. In *AAAI*, 2017.

[27] A. Gupta, V. Kumar, C. Lynch, S. Levine, and K. Hausman. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. *CoRL*, 2019.

[28] A. Mandlekar, F. Ramos, B. Boots, L. Fei-Fei, A. Garg, and D. Fox. Iris: Implicit reinforcement without interaction at scale for learning control from offline robot manipulation data. *ICRA*, 2020.

[29] Y. Lee, S.-H. Sun, S. Somasundaram, E. S. Hu, and J. J. Lim. Composing complex skills by learning transition policies. In *International Conference on Learning Representations*, 2018.

[30] T. Kipf, Y. Li, H. Dai, V. Zambaldi, E. Grefenstette, P. Kohli, and P. Battaglia. Compositional imitation learning: Explaining and executing one task at a time. *ICML*, 2019.

[31] W. Whitney, R. Agarwal, K. Cho, and A. Gupta. Dynamics-aware embeddings. *ICLR*, 2020.

[32] N. Y. Siegel, J. T. Springenberg, F. Berkenkamp, A. Abdolmaleki, M. Neunert, T. Lampe, R. Hafner, and M. Riedmiller. Keep doing what worked: Behavioral modelling priors for offline reinforcement learning. *ICLR*, 2020.

[33] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

[34] C. Gulcehre, Z. Wang, A. Novikov, T. L. Paine, S. G. Colmenarejo, K. Zolna, R. Agarwal, J. Merel, D. Mankowitz, C. Paduraru, et al. Rl unplugged: Benchmarks for offline reinforcement learning. *arXiv preprint arXiv:2006.13888*, 2020.

[35] S. Schaal, J. Peters, J. Nakanishi, and A. Ijspeert. Learning movement primitives. In P. Dario and R. Chatila, editors, *Robotics Research*. Springer Berlin Heidelberg, 2005.

[36] A. Mandlekar, Y. Zhu, A. Garg, J. Booher, M. Spero, A. Tung, J. Gao, J. Emmons, A. Gupta, E. Orbay, S. Savarese, and L. Fei-Fei. Roboturk: A crowdsourcing platform for robotic skill learning through imitation. In *Conference on Robot Learning*, 2018.

[37] K. Fang, Y. Zhu, A. Garg, S. Savarese, and L. Fei-Fei. Dynamics learning with cascaded variational inference for multi-step manipulation. *CoRL 2019*, 2019.

[38] K. Pertsch, O. Rybkin, J. Yang, S. Zhou, K. Derpanis, J. Lim, K. Daniilidis, and A. Jaegle. Keyframing the future: Keyframe discovery for visual prediction and planning. *L4DC*, 2020.

[39] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner. beta-VAE: Learning basic visual concepts with a constrained variational framework. In *ICLR*, 2017.

[40] D. P. Kingma and M. Welling. Auto-encoding variational Bayes. In *ICLR*, 2014.

[41] D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *ICML*, 2014.

[42] C. M. Bishop. *Pattern recognition and machine learning*. springer, 2006.

[43] D. J. Rezende and S. Mohamed. Variational inference with normalizing flows. *ICML*, 2015.

[44] L. Dinh, J. Sohl-Dickstein, and S. Bengio. Density estimation using real nvp. *ICLR*, 2017.

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

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

[47] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *ICML*, 2018.

[48] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[49] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

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. [47] and Levine [46] 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 [46], 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:

$$p(\tau|\mathcal{O}_{1:T}) = p(s_1) \prod_{t=1}^{T} p(\mathcal{O}_t|s_t, a_t) p(s_{t+1}|s_t, a_t) p(a_t|\cdot)$$

$$= \left[p(s_1) \prod_{t=1}^{T} p(s_{t+1}|s_t, a_t) p(a_t|\cdot) \right] \cdot \exp \sum_{t=1}^{T} r(s_t, a_t)$$

Crucially, in contrast to [46] 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 [46] 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:

$$\log p(\mathcal{O}_{1:T}) \geq -D_{\text{KL}}\big[q(\tau) \,||\, p(\tau|\mathcal{O}_{1:T})\big]$$

$$\geq \mathbb{E}_{\tau \sim q(\tau)} \left[\log p(s_1) + \sum_{t=1}^{T} \big[\log p(s_{t+1}|s_t, a_t) \log p(a_t|\cdot) \big] + \sum_{t=1}^{T} r(s_t, a_t) \right.$$

$$\left. - \log p(s_1) - \sum_{t=1}^{T} \big[\log p(s_{t+1}|s_t, a_t) \log q(a_t|s_t) \big] \right]$$

$$\geq \mathbb{E}_{\tau \sim q(\tau)} \left[\sum_{t=1}^{T} r(s_t, a_t) + \log p(a_t|\cdot) - \log q(a_t|s_t) \right]$$

$$\geq \mathbb{E}_{\tau \sim q(\tau)} \left[\sum_{t=1}^{T} r(s_t, a_t) - D_{\text{KL}}\big[q(a_t|s_t) \,||\, p(a_t|\cdot)\big] \right]$$

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 [46].

To maximize this ELBO with respect to the policy $q(a_t|s_t)$, [46] 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 [46], 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:

$$\mathcal{T}^\pi Q(s_t, a_t) = r(s_t, a_t) + \gamma \mathbb{E}_{s_{t+1} \sim p}\big[V(s_{t+1})\big]$$

$$\text{where } V(s_t) = \mathbb{E}_{a_t \sim \pi}\big[Q(s_t, a_t) - D_{\text{KL}}\big[\pi(a_t|s_t) \,||\, p(a_t|\cdot)\big]\big]$$

To show convergence of this operator to the optimal Q function we follow the proof of [47] in appendix B1 and introduce a divergence-augmented reward:

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

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 [49].

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

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

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. [48] 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:

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

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:

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

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

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

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   Implementation Details

B.1   Model Architecture and Training Objective

We instantiate the skill embedding model described in Section 3.2 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 $1e-3$. Training on a single NVIDIA Titan X GPU takes approximately 8 hours. Assuming a unit-variance Gaussian output distribution our full training objective is:

$$\mathcal{L} = \sum_{i=1}^{H} \underbrace{(a_i - \hat{a}_i)^2}_{\text{reconstruction}} - \beta \underbrace{D_{\mathrm{KL}} \big(\mathcal{N}(\mu_z, \sigma_z) || \mathcal{N}(0, I) \big)}_{\text{regularization}} + \underbrace{D_{\mathrm{KL}} \big(\mathcal{N}(\lfloor \mu_z \rfloor, \lfloor \sigma_z \rfloor) || \mathcal{N}(\mu_p, \sigma_p) \big)}_{\text{prior training}}. \quad (4)$$

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 $\beta = 1e-2$ for maze and block stacking and $\beta = 5e-4$ for the kitchen environment.

B.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 (see Section F).

We use the hyperparameters of the standard SAC implementation [47] with batch size 256, replay buffer capacity of $1e6$ and discount factor $\gamma = 0.99$. We collect 5000 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 $3e{-}4$ 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 $\tau = 5e{-}3$. The policy's actions are limited in the range $[-2, 2]$ by a $\tanh$ "squashing function" (see Haarnoja et al. [47], 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.

C Environments and Data Collection

Maze Navigation. The maze navigation environment is based on the maze environment in the D4RL benchmark [33]. 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. [33], 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.
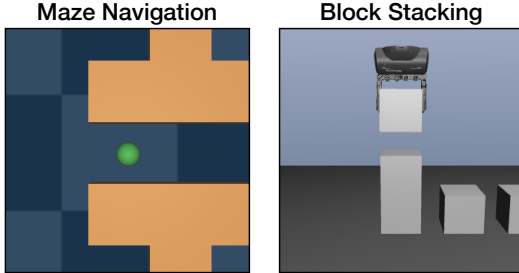

Figure 7: Image-based state representation for maze (**left**) and block stacking (**right**) environment (downsampled to 32×32 px for policy).

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. 7). 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. 7). 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 $[0 \ldots 1]$ 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 [33] which was originally published by Gupta et al. [27]. For training we use the data provided in D4RL (dataset version "mixed"). It consists of trajectories collected via human tele-operation that each perform
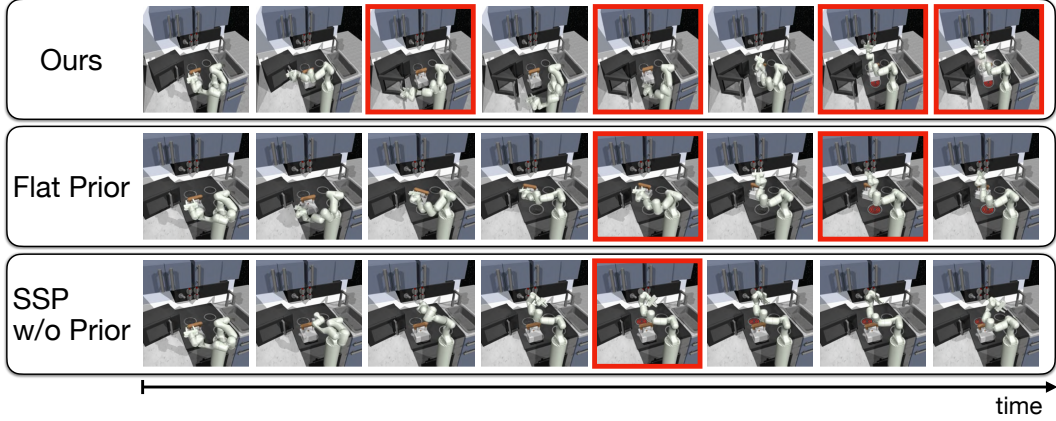
Figure 8: Comparison of policy execution traces on the kitchen environment. Following Fu et al. [33], the agent's task is to (1) open the microwave, (2) move the kettle backwards, (3) turn on the burner and (4) switch on the light. Red frames mark the completion of subtasks. Our skill-prior guided agent (**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. For videos, see: `clvrai.com/spirl`.
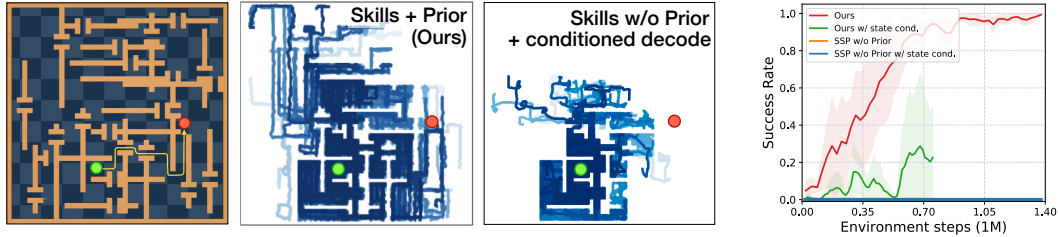

Figure 9: Results for state-conditioned skill decoder network. **left**: Exploration visualization as in Fig. 5. Even with state-conditioned skill decoder, exploration without skill prior is not able to explore a large fraction of the maze. In contrast, skills sampled from the learned skill prior lead to wide-ranging exploration when using the state-conditioned decoder. **right**: Downstream learning performance of our approach and skill-space policy w/o learned skill prior: w/ vs. w/o state-conditioning for skill decoder. Only guidance through the learned skill prior enables learning success. State-conditioned skill-decoder can make the downstream learning problem more challenging, leading to lower performance ("ours" vs. "ours w/ state cond.").

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.

D State-Conditioned Skill Decoder

Following prior works [7, 30], we experimented with conditioning the skill decoder on the current environment state s_1. Specifically, the current environment state is passed through a neural network
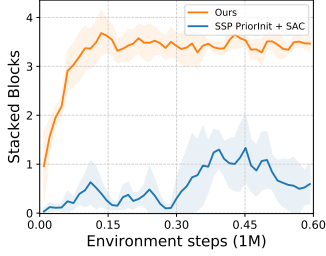
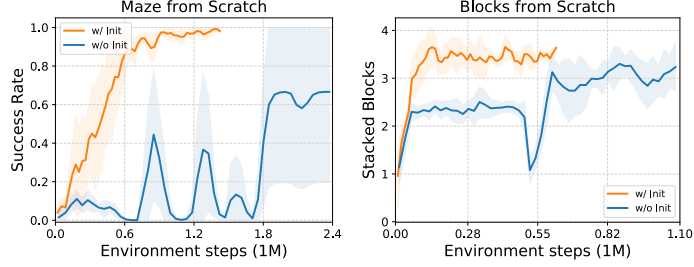|	
Figure 10: Ablation of prior regularization during downstream RL training. Initializing the high-level policy with the learned prior but finetuning with conventional SAC is not sufficient to learn the task well.	Figure 11: Ablation of prior initialization. Initializing the downstream task policy with the prior network improves training stability and convergence speed. However, the "w/o Init" runs demonstrate that the tasks can also be learned with prior regularization only.

that predicts the initial hidden state of the decoding LSTM. We found that conditioning the skill decoder on the state did not improve downstream learning and can even lead to worse performance. In particular, it does not improve the exploration behavior of the skill-space policy *without* learned prior (see Fig. 9, left); a learned skill prior is still necessary for efficient exploration on the downstream task.

Additionally, we found that conditioning the skill decoder on the state can reduce downstream learning performance (see Fig. 9, right). We hypothesize that state-conditioning can make the learning problem for the high-level policy more challenging: due to the state-conditioning the same high-level action z can result in different decoded action sequences depending on the current state, making the high-level policies' action space dynamics more complex. As a result, downstream learning can be less stable.

E   Prior Regularization Ablation

We ablate the influence of the prior regularization during downstream learning as described in Section 3.3. Specifically, we compare to an approach that initializes the high-level policy with the learned skill prior, but then uses conventional SAC [47] (with uniform skill prior) to finetune on the downstream task. Fig. 10 shows that the prior regularization during downstream learning is essential for good performance: conventional maximum-entropy SAC ("SSP PriorInit + SAC") quickly leads the prior-initialized policy to deviate from the learned skill prior by encouraging it to maximize the entropy of the distribution over skills, slowing the learning substantially.

F   Prior Initialization Ablation

For the RL experiments in Section 4 we initialize the weights of the high-level policy with the learned skill prior network. Here, we compare the performance of this approach to an ablation that does not perform the initialization. We find that prior initialization improves convergence speed and stability of training (see Fig. 11).

We identify two major challenges that make training policies "from scratch", without initialization, challenging: (1) sampling-based divergence estimates between the randomly initialized policy and the prior distribution can be inaccurate in early phases of training, (2) learning gets stuck in local optima where the divergence between policy and prior is minimized on a small subset of the state space, but the policy does not explore beyond this region.

Since both our learned prior and the high-level policy are parametrized with Gaussian output distributions, we can analytically compute the KL divergence to get a more stable estimate and alleviate problem (1). To address problem (2) when training from scratch, we encourage exploration by sampling a fraction ω of the rollout steps during experience collection directly from the prior instead of the policy. For the "w/o Init" experiments in Fig. 11 we set $\omega = 1.0$ for the first $500\,\mathrm{k}$ steps (i.e.

always sample from the prior) and then anneal it to 0 (i.e. always use the policy for rollout collection) over the next $500\,\mathrm{k}$ steps. Note that policies trained with prior initialization do not require these additions and still converge faster.

G   Training with Sub-Optimal Data

We investigate the influence of the training data on the performance of our approach. In particular, we test whether it is possible to learn effective skill priors from heavily sub-optimal data.

Table 1: Number of blocks stacked vs fractions of random training data.

% Random Data	0%	50%	75%
# Blocks Stacked	3.5	2.0	1.0

For the experiments in Section 4 we trained the skill prior from high-quality experience collected using expert-like policies, albeit on tasks that differ from the downstream task (see Section C). However, in many practical cases the quality of training data can be mixed. In this Section we investigate two scenarios for training from sub-optimal data.

Mixed Expert and Random Data.   We assume that a large fraction of the data is collected by inexperienced users leading to very low quality trajectories, while another part of the data is collected by experts and has high quality. We emulate this in the block stacking environment by combining rollouts collected by executing random actions with parts of the high-quality rollouts we used for the experiments in Section 4.3.

The results are shown in table 1. Our approach achieves good performance even when half of the training data consists of very low quality rollouts, and can learn meaningful skills for stacking blocks when $75\,\%$ of the data is of low quality. The best baseline was only able to stack 1.5 blocks on average, even though it was trained from only high-quality data (see Fig. 4, middle).

Only Non-Expert Data.   We assume access to a dataset of only mediocre quality demonstrations, without any expert-level trajectories. We generate this dataset in the maze environment by training a behavior cloning (BC) policy on expert-level trajectories and using it to collect a new dataset. Due to the limited capacity of the BC policy, this dataset is of substantially lower quality than the expert dataset, e.g. the agent collides with walls on average 2.5 times per trajectory while it never collides in expert trajectories.
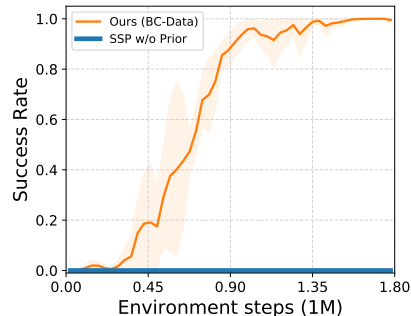

Figure 12: Success rate on maze environment with sub-optimal training data. Our approach, using a prior learned from sub-optimal data generated with the BC policy, is able to reliably learn to reach the goal while the baseline that does not use the learned prior fails.

While we find that a skill-prior-regularized agent trained on the mediocre data explores the maze less widely than one trained on the expert data, it still works substantially better than the baseline that does not use the skill prior, achieving $100\,\%$ success rate of reaching a faraway goal after <1M environment steps, while the baseline does not reach the goal even after 3M environment steps.

Both scenarios show that our approach can learn effective skill embeddings and skill priors even from substantially sub-optimal data.

H   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. 3 (left) and use it to train multiple downstream task agents that reach different goals.
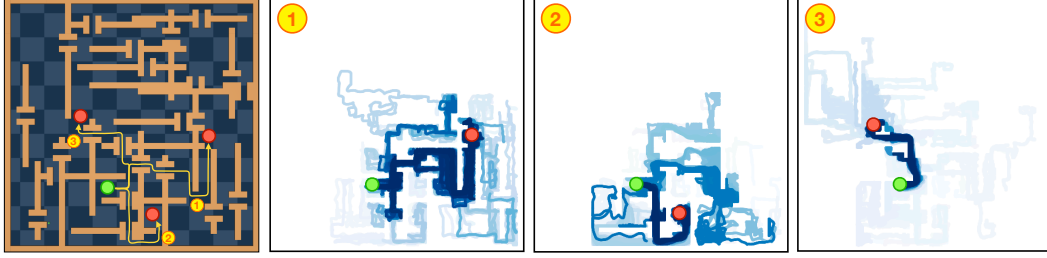
16

Figure 13: 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. 13 we show a visualization of the training rollouts in a top-down view, similar to the visualization in Fig. 5; 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 bumping into walls, which helps exploration in the maze independent of the goal position.