S1 Abstract

Skill-based reinforcement learning (Skill-based RL) is an efficient paradigm for solving sparse-reward tasks by extracting skills from demonstration datasets and learning high-level policy which selects skills. Because each selected skill by high-level policy is executed for multiple consecutive timesteps, the high-level policy is essentially learned in a temporally abstract Markov decision process (TA-MDP) built on skills, which shortens task horizons and reduces exploration costs. Yet, these skills are usually sub-optimal because of the limited quality and coverage of datasets, which causes the sub-optimal performance in downstream tasks. Refining skills is intuitive, but ensuring performance improvement meanwhile is a hard issue. To address the dilemma of sub-optimality and ineffectiveness, we propose a unified optimization objective for the entire hierarchical policy. We prove that this objective guarantees the performance improvement in TA-MDP, and optimizing the performance in TA-MDP can be equivalent to optimizing the performance lower bound in original MDP. Furthermore, in order to overcome the phenomenon of skill space collapse, we propose the dynamical skill refinement (DSR) which names our method. The experiment results empirically validate the effectiveness of our method, and show the advantages over the state-of-the-art (SOTA) methods.

S2 Introduction

Reinforcement learning has been widely used in complex application scenarios, such as robotics manipulation [1–8]. These tasks are usually with sparse reward functions which give a positive feedback only when the task is completed. For reducing the extremely high exploration costs caused by sparse reward functions, prior works have proposed a variety of methods to incorporate prior knowledge into reinforcement learning [4, 5, 8–13]. In these works, skill-based reinforcement learning (Skill-based RL) has become an efficient approach [3, 5, 8]. Skills are temporally extended behaviors and usually manifest as the segments of consecutive actions in demonstration datasets. These skills are typically embedded into the latent space of the low-level policy which serves as the action space of the high-level policy. The high-level policy selects the appropriate skills according to the states, and lets each selected skill be executed for multiple consecutive timesteps so as to shorten task horizons and reduce exploration costs.

These skills may be sub-optimal because of the limited quality and coverage of datasets [14], which causes the potential sub-optimal performance in downstream tasks. It is intuitive to refine the extracted skills with online collected transitions while learning the high-level policy. However, refining skills while ensuring performance improvement remains a challenging issue. Firstly, changes of skills lead to the non-stationarity of the transition dynamics of the temporally abstract Markov decision process (TA-MDP), which we name temporal abstraction shift. Temporal abstraction shift makes it difficult to accurately estimate the values of skills. Furthermore, prior works lack a unified optimization objective that can theoretically guarantees performance improvement.

Therefore, we have to face the dilemma consisting of sub-optimal skills and ineffective skill refinements. Some prior methods assume that skills are nearly optimal and keep them fixed, but this assumption puts high requirements on datasets, which can hardly be met when the downstream task is different from all the tasks used to generate the datasets [3, 5, 8]. Skill-Critic [15] directly ignores the temporal abstraction shift and updates both the high-level policy and the skills in the off-policy RL manner [16]. Skill-Critic achieves superior performance and sample efficiency in some specific tasks. However, ignoring the temporal abstraction shift brings in uncertainty, and Skill-Critic depends on SPiRL-based [5] warm-up stage. ReSkill [17] updates both the high-level policy and the skills in an on-policy RL manner [18, 19], which circumvents the temporal abstraction shift. However, in ReSkill, the high-level policy and the skills are updated in the TA-MDP and the original MDP respectively. Inconsistent optimization objectives can not theoretically guarantee the performance improvement.

In order to address the dilemma, we propose an on-policy skill-based RL method named dynamical skill refinement (DSR) which dynamically refines the skills under the optimization objective unified with the high-level policy. We theoretically prove that this objective guarantees the performance improvement in TA-MDP. Furthermore, we theoretically prove that optimizing the performance in TA-MDP can be equivalent to optimizing the performance lower bound in original MDP. Therefore, we can implement performance improvement with this objective in a lower-bound optimization. Additionally, we point out that directly refining skills may lead to skill space collapse which brings in performance collapse. This is because all skills are embedded into the latent space of the same parametric low-level policy. Changing the behavior of one skill may cause the behaviors of other skills to be changed as well. Therefore, we augment our method with the dynamical skill refinement mechanism which ensures that the skills maintain their original behaviors before being explored. We empirically validate the effectiveness of our method and its advantages over the state-of-the-art (SOTA) methods in multiple sparse-reward tasks of the robotics manipulation domain, and prove the necessity of dynamical skill refinement.

We summarize the contributions of this paper as follows: (1) we refine the skills in a manner of performance lower bound optimization, so as to ensure performance improvement, (2) we devise an on-policy skill-based RL method and a dynamical

skill refinement mechanism which avoids skill space collapse, (3) we validate the effectiveness and superiority of our method in several sparse-reward tasks.

S3 Related Work

S3.1 Hierarchical Reinforcement Learning

In hierarchical reinforcement learning (HRL), the policy stitches temporally extended behaviors rather than primitive actions to be the behavior of solving tasks [3, 20–27]. The low-level policy is usually conditioned on the latent space which serves as the action space of the high-level policy. Both the high-level and the low-level policies can be learned from the experience obtained from the agent's interacting with the environment [28–30]. In practical applications, it is a mainstream approach to extract the low-level policy from the demonstration dataset and then learn to recompose the behavior modes of the low-level policy to solve the task [3, 31–33].

S3.2 Skill-based Reinforcement Learning

Skill-based reinforcement learning (skill-based RL) extracts reusable behavior modes from the demonstration dataset which are used to solve the downstream task [5,8,34,35]. These behavior modes are considered as skills and embedded into the latent space, which is typically through a variational auto-encoder (VAE) [36]. During online learning, the high-level policy learns to select the optimal skills to solve the downstream task. The high-level policy learns to choose temporally extended skills rather than single-timestep actions, which shortens the task horizon and reduces the exploration cost. Prior works generally focus on extracting the skill prior from the demonstrations to improve the exploration efficiency [5, 8, 37–39]. These works assume that the skills can be approximately considered as optimal for the downstream task. However, this assumption can hardly be met when the downstream task is different from the tasks used to generate the demonstration dataset. It is intuitive and natural to refine the skills while learning the high-level policy [15, 17], but this issue is still not well addressed.

S4 Preliminary

S4.1 TA-MDP

In skill-based RL, the high-level policy π^h and the low-level policy π^l whose latent space embeds the skills $\{z\}$ constitute the entire hierarchical policy. The high-level policy selects the skills from the latent space Z which are the behavior modes of the low-level policy π^l conditioned on the latent variables. Each skill manifests as a sequence of primitive actions $\{a_t, a_{t+1}, ..., a_{t+H-1}\}$ over a fixed horizon H. Once a skill z_t is selected from $\pi^h(\cdot|s_t)$ by the high-level policy, the action distributions at the next H consecutive timesteps will be $a_{t+i} \sim \pi^l(\cdot|s_{t+i}, z_t), i \in [0, H-1]$.

In reinforcement learning, the task is formulated in a Markov decision process (MDP) $(\mathcal{S}, \mathcal{A}, p, r, \gamma)$ which consists of state space $\mathcal{S}$, action space $\mathcal{A}$, transition dynamics $p(s'|s, a)$, reward function $r(s, a)$ and discount factor γ. In skill-based RL, each skill z selected by high-level policy π^h is over a fixed horizon H, so π^h is actually learning in a TA-MDP $(\mathcal{S}, \mathcal{Z}, p_{\pi^l,H}, \tilde{r}, \tilde{\gamma})$. TA-MDP's action space $\mathcal{Z}$ is the latent space of low-level policy $\pi^l(\cdot|s, z)$. TA-MDP's transition dynamics $p_{\pi^l,H}(s'|s, z)$ is the distribution over the state s_{t+H} after H timesteps conditioned on the state s_t and the selected skill z_t, $p(s_{t+H} = s'|s_t = s, z_t = z, \pi^l)$. TA-MDP's reward function $\tilde{r}(s_t, z_t) = \sum_{i=0}^{H-1} r_{t+i}$ is the sum of H consecutive rewards obtained by executing skill z_t from state s_t. $\tilde{\gamma}$ is the discount factor.

S4.2 Extract Skills through VAE

In order to embed the skills into the latent space of low-level policy, it is common to sample state-action segments of fixed length from the demonstration dataset and encode them as latent variables through the variational auto-encoder (VAE) [5,8,17]. The decoder conditioned on state and latent variable will reconstruct action segments. The sum of the loss of reconstructed action segment and the KL-divergence between the latent variable distribution and a prior distribution is the complete optimization objective [5,8,17,25]. The trained decoder will serve as the low-level policy. Usually, a skill prior that is conditioned on the first state and attempts to predict the same latent variable will also be learned to improve the exploration efficiency in the online learning stage. The skill prior can be used as the initial high-level policy and be used for biasing explorations to the regions of potentially valuable skills. The extraction process is sketched in Fig. S1.

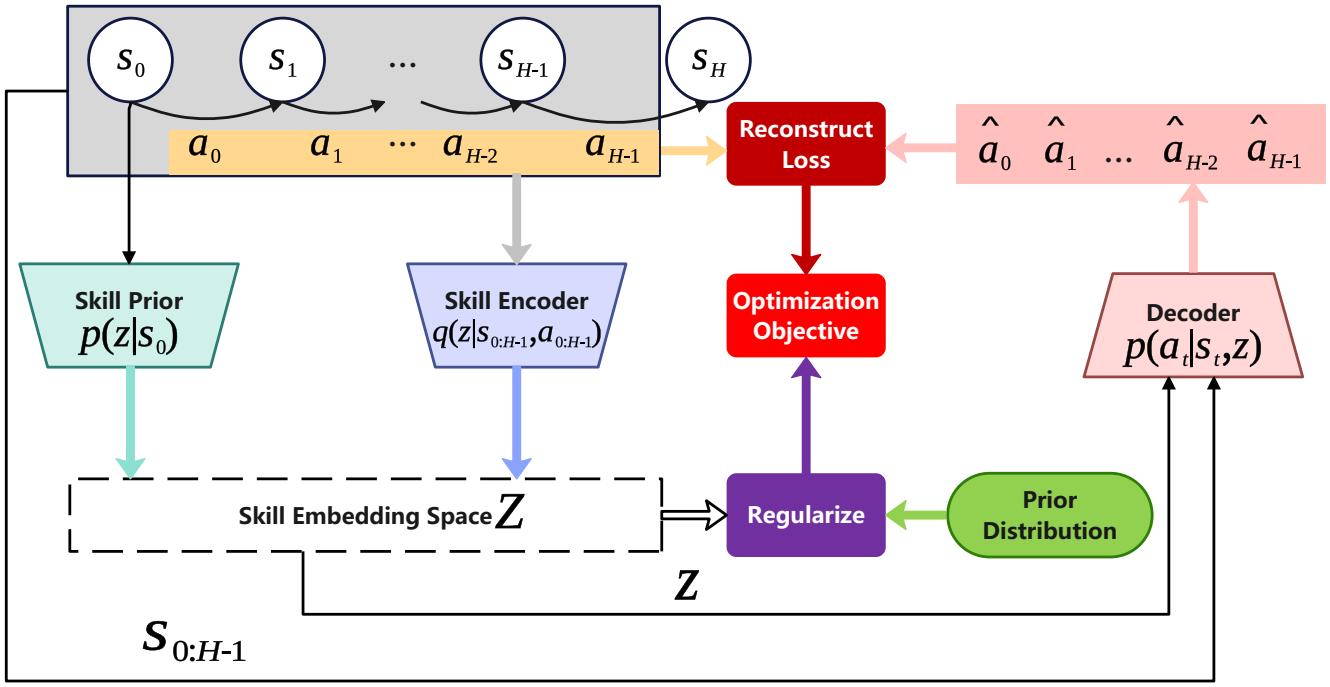


Fig. S1 State-action segments of fixed length are sampled from the demonstration dataset. These segments are embedded into the latent space by the auto-encoder. The decoder reconstructs the action segment based on the state segment and the latent variable. We not only reduce the reconstruction loss, but also make the distribution of skill embedding close to the prior distribution. Skill prior learns to predict the skill embedding of the state-action segment based on only the first state.

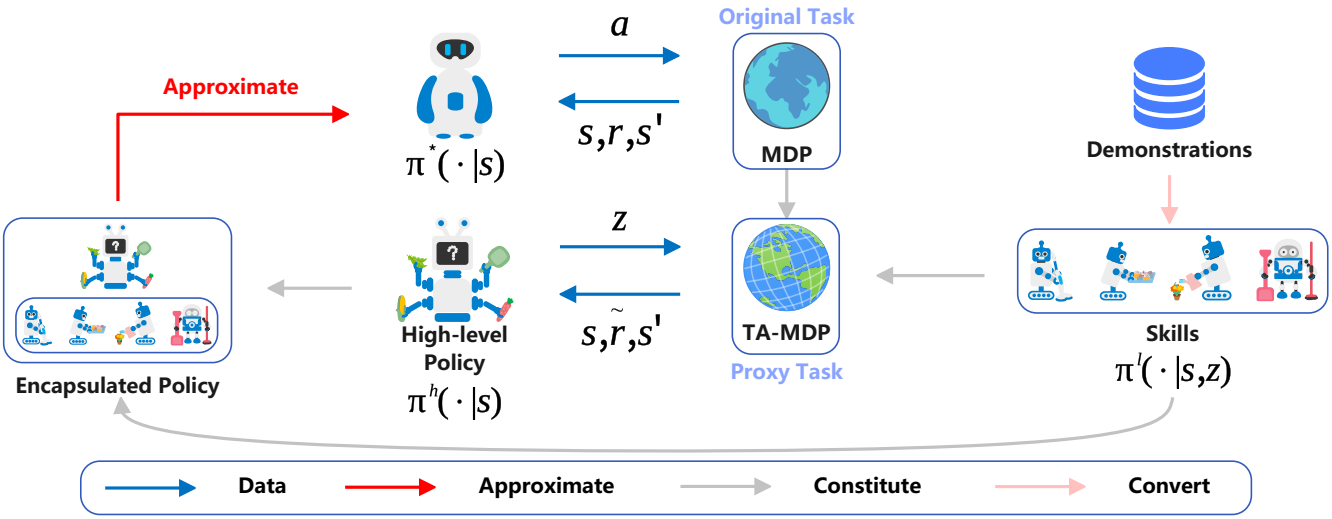


Fig. S2 The extracted skills and the original MDP constitute the TA-MDP. We regard learning the hierarchical policy in TA-MDP as a proxy task of learning a flat policy in the original MDP. We consider the hierarchical policy as a whole, which gives an action at each timestep, just like the flat policy.

S5 Update High-level Policy and Skills under Unified Optimization Objective

We take the expected sum of discounted rewards in TA-MDP as the unified optimization objective of both high-level policy and skills. Specifically, the optimization objective of skills is the sum of inner-skill single-timestep MDP rewards and the subsequent discounted H-timestep TA-MDP rewards. We prove that this optimization objective ensures the performance improvement in TA-MDP. Furthermore, we innovatively consider skill-based RL as a proxy task of RL, and prove that optimizing the expected sum of discounted H-timestep TA-MDP rewards is equivalent to optimizing a lower bound of the expected sum

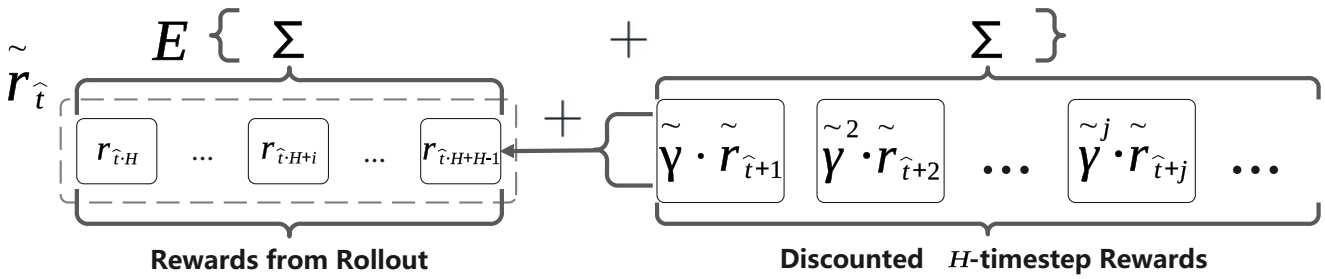


Fig. S3 The H consecutive transitions generated by the same skill can be seen as a rollout. We add the sum of discounted H-timestep rewards $\tilde{r}$ to the last reward $r_{\hat{t}\cdot H+H-1}$ in it. Then, we apply on-policy RL methods on the resulted rollout to optimize the skill. In this way, we equivalently optimize the expected sum of inner-skill rewards and discounted H-timestep rewards after the skill.

of the discounted single-timestep MDP rewards, which provides the effectiveness.

S5.1 Optimization Objective and Update Formulae

In skill-based RL, the task of high-level policy is to maximize the expected sum of discounted H-timestep TA-MDP rewards $\tilde{r}(s,z)$, which can be formulated as follows:

$$\max_{\pi^h} \mathbb{E}_{\pi^h,\pi^l}[\sum_{\hat{t}} \tilde{\gamma}^{\hat{t}} \cdot \tilde{r}_{\hat{t}}], \text{where } \tilde{r}_{\hat{t}} = \sum_{t=\hat{t}\cdot H}^{\hat{t}\cdot H+H-1} r_t. \tag{S1}$$

$\hat{t}$ and t are the timestep index variables of TA-MDP and MDP respectively. Accordingly, we design the optimization objective of skills to be the sum of all the inner-skill single-timestep rewards and the following discounted H-timestep rewards. We formulate this objective as follows:

$$\max_{\pi^l} \mathbb{E}_{\pi^h,\pi^l}[\sum_{i=\hat{t}\cdot H}^{\hat{t}\cdot H+H-1} r_i + \sum_{j=\hat{t}+1}^{\infty} \tilde{\gamma}^{j-\hat{t}} \cdot \tilde{r}_j]. \tag{S2}$$

To formulate policy iteration, we define state value function $V^h_{\pi^h,\pi^l}$ and state-skill value function $Q^h_{\pi^h,\pi^l}$ in TA-MDP:

$$V^h_{\pi^h,\pi^l}(s) = \mathbb{E}_{\pi^h,\pi^l,s_0=s}[\sum_{\hat{t}=0} \tilde{\gamma}^{\hat{t}} \cdot \tilde{r}_{\hat{t}}], \tag{S3}$$

$$Q^h_{\pi^h,\pi^l}(s,z) = \mathbb{E}_{a_t\sim\pi^l(\cdot|s_t,z),s_0=s}[\sum_{t=0}^{H-1} r_t] + \mathbb{E}_{s'\sim p_{\pi^l,H}(\cdot|s,z)}[V^h_{\pi^h,\pi^l}(s')], \tag{S4}$$

where $p_{\pi^l,H}(\cdot|s,z)$ is the distribution of reached state after executing z from s for H timesteps.

Therefore, the policy-iteration optimization objective of the high-level policy regarding a state s is as follows:

$$\max_{\pi^h} \mathbb{E}_{z\sim\pi^h(\cdot|s)}[Q^h_{\pi^h,\pi^l}(s,z)]. \tag{S5}$$

The policy-iteration optimization objective of the skill z executed from state s at timestep $\hat{t}\cdot H$ is as follows:

$$\max_{\pi^l}[\mathbb{E}_{a_t\sim\pi^l(\cdot|s_t,z),s_{\hat{t}\cdot H}=s}[\sum_{t=\hat{t}\cdot H}^{\hat{t}\cdot H+H-1} r_t] + \tilde{\gamma}\cdot\mathbb{E}_{s'\sim p_{\pi^l,H}(\cdot|s,z)}[V^h_{\pi^h,\pi^l}(s')]], \tag{S6}$$

We parameterize π^h and π^l as π^h_ϕ and π^l_θ. Then, the recursive update formulae for π^h_ϕ and π^l_θ are as follows:

$$\forall s, \pi^h_{\phi'}(\cdot|s) \leftarrow \arg\max_{\pi^h(\cdot|s)} \mathbb{E}_{z\sim\pi^h(\cdot|s)}[Q^h_{\pi^h_\phi,\pi^l_\theta}(s,z)], \tag{S7}$$

$$\forall s_{\hat{t}\cdot H}, \forall z, \pi^l_{\theta'}(\cdot|s_t,z) \leftarrow \arg\max_{\pi^l(\cdot|s_t,z)}[\mathbb{E}_{a_t\sim\pi^l(\cdot|s_t,z),s_{\hat{t}\cdot H}}[\sum_{t=\hat{t}\cdot H}^{\hat{t}\cdot H+H-1} r_t] + \tilde{\gamma}\cdot\mathbb{E}_{s'\sim p_{\pi^l,H}(\cdot|s_{\hat{t}\cdot H},z)}[V^h_{\pi^h_\phi,\pi^l_\theta}(s')]], \tag{S8}$$

ϕ' and θ' are the updated version of parameters ϕ and θ.

S5.2 Performance Improvement in TA-MDP

We prove that the updates of π^h_ϕ, π^l_θ following Eq. S7 and Eq. S8 guarantee the performance improvement in TA-MDP and the performance in TA-MDP will converge.

To prove that there is a monotonic increase in the state value function, we first illustrate Lemma 1:

Lemma 1. $\forall s \in \mathcal{S}$, we denote the value of first predicting a skill z with $\pi^h_{\phi'}$, executing $\pi^l_{\theta'}(\cdot|s,z)$ for H timesteps and then executing π^h_ϕ, π^l_θ by $\tilde{V}^h_{\pi^h_{\phi'},\pi^l_{\theta'},\pi^h_\phi,\pi^l_\theta}(s)$. The following inequality holds, if we follow the update formulae in Eq. S7 and Eq. S8:

$$\forall s \in \mathcal{S}, \tilde{V}^h_{\pi^h_{\phi'},\pi^l_{\theta'},\pi^h_\phi,\pi^l_\theta}(s) \geq V^h_{\pi^h_\phi,\pi^l_\theta}(s). \tag{S9}$$

See Appendix SA.1 for proof.

Based on Lemma 1, we prove the monotonic performance increase in the TA-MDP:

Theorem 1. If we update π^h_ϕ and π^l_θ following Eq. S7 and Eq. S8, then:

$$\forall s \in \mathcal{S}, V^h_{\pi^h_{\phi'},\pi^l_{\theta'}}(s) \geq V^h_{\pi^h_\phi,\pi^l_\theta}(s). \tag{S10}$$

See Appendix SA.2 for proof.

Finally, we get the convergence of performance:

Theorem 2. With the policy update formulae in Eq. S7 and Eq. S8, the state value function $V^h_{\pi^h_{\phi_k},\pi^l_{\theta_k}}(s)$ will converge. $\pi^h_{\phi_k}, \pi^l_{\theta_k}$ are the k-th version of high-level policy and low-level policy respectively. See Appendix SA.3 for proof.

S5.3 Effectiveness of Learning in TA-MDP

To analyze the effectiveness, we propose a novel perspective that the essential purpose of skill-based RL should be to make the behavior of the entire hierarchical policy approximate to that of the optimal policy in original MDP. Unlike prior works [5, 8, 15, 17] which considers skill-based RL as a paradigm parallel to RL, our perspective establishes a connection between skill-based RL and RL. We sketch it in Fig. S2 which motivates us to analyze the relationship between the performance in TA-MDP and that in original MDP. We prove that optimizing the performance of a hierarchical policy in TA-MDP is equivalent to optimizing a lower bound of its performance in original MDP.

Given MDP $(\mathcal{S}, \mathcal{A}, p, r, \gamma)$ and hierarchical policy (π^h, π^l), we denote the TA-MDP by $(\mathcal{S}, \mathcal{Z}, p_{\pi^l,H}, \tilde{r}, \tilde{\gamma})$. The performance of (π^h, π^l) in original MDP and that in TA-MDP are expanded into analogous forms:

$$V_{\pi^h,\pi^l}(s) = \int \sum_{\triangle t=0}^{\infty} \rho_{\triangle t}(s', a'|s, \pi^h, \pi^l)\gamma^{\triangle t} r(s', a') da' ds', \tag{S11}$$

$$V^h_{\pi^h,\pi^l}(s) = \int \sum_{\triangle t=0}^{\infty} \rho_{\triangle t}(s', a'|s, \pi^h, \pi^l)\tilde{\gamma}^{\lfloor \triangle t/H \rfloor} r(s', a') da' ds', \tag{S12}$$

$\rho_{\triangle t}(s', a'|s, \pi^h, \pi^l)$ is the state-action distribution after executing π^h, π^l for $\triangle t$ timesteps from s.

Since the discount factor is manually specified, we can let $\tilde{\gamma} = \gamma^H$. We illustrate in Theorem 3 that optimizing the performance of the hierarchical policy in TA-MDP is equivalent to optimizing its performance lower bound in original MDP if the reward function gives a positive feedback only when the task is completed.

Theorem 3. If the MDP $(\mathcal{S}, \mathcal{A}, p, r, \gamma)$ and the TA-MDP $(\mathcal{S}, \mathcal{Z}, p_{\pi^l,H}, \tilde{r}, \tilde{\gamma})$ satisfy that $\tilde{\gamma} = \gamma^H$, then $\tilde{\gamma} \cdot V^h_{\pi^h,\pi^l}(s) \leq V_{\pi^h,\pi^l}(s)$ holds for $\forall s \in \mathcal{S}$. It means that optimizing $V^h_{\pi^h,\pi^l}(s)$ is equivalent to optimizing a lower bound of $V_{\pi^h,\pi^l}(s)$. See Appendix SA.4 for proof.

Since we have proved in Theorem 1 that our optimization objective guarantees the monotonic increase in $V^h_{\pi^h,\pi^l}(s)$, refining skills under this objective is equivalent to optimizing the lower bound of $V_{\pi^h,\pi^l}(s)$, which illustrates the effectiveness of our method.

S6 Update Hierarchical Policy with Dynamical Skill Refinement in an On-policy Manner

We first illustrate how to optimize our objective in an on-policy RL manner. Then, we elaborate on the dynamical skill refinement mechanism and explain its necessity.

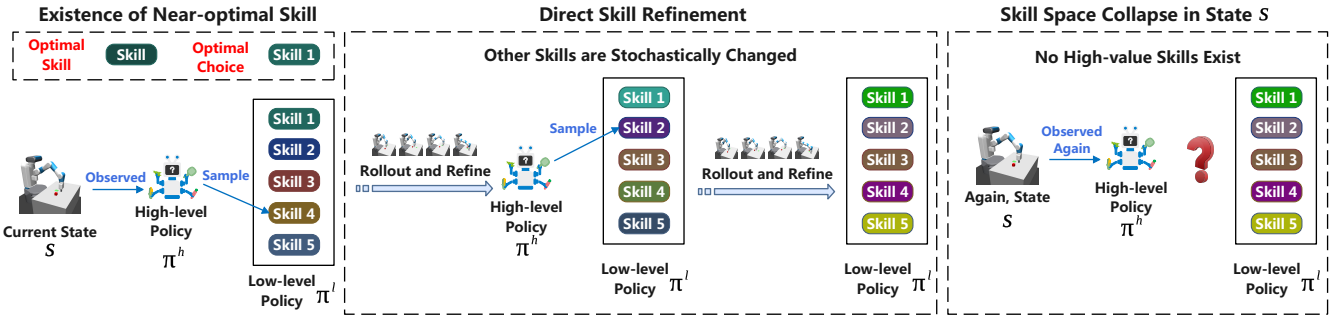


Fig. S4 Before predicting the optimal skill, the high-level policy usually needs to sufficiently explore the skill space in a given state. If we refine selected skills directly, a terrible selected skill will not quickly approach the optimal skill, but other skills will change stochastically. The best skill in the skill library for a state may be destroyed after consecutive rollouts and skill refinements. When encountering the state again, there are actually no good candidate skills that can be explored.

S6.1 Optimizing the Objective in an On-policy RL Manner

Since high-level policy is learned in TA-MDP, we can directly apply the on-policy RL algorithms [18,19] to learn it. Therefore, we focus on illustrating how to refine the skills.

We first observe that the policy-iteration optimization objective of skills in Eq. S8 can be decomposed into two components: (1) the expected sum of the inner-skill rewards given by original MDP and (2) the expected sum of the discounted H-timestep rewards given by TA-MDP. In addition, a trajectory is composed of transition segments with the length of H timesteps generated by multiple skills. Therefore, we innovatively consider the H-timestep transition segment generated by a skill in trajectories as a rollout and equivalently add the sum of subsequent discounted H-timestep rewards to the last reward. This idea is sketched in Fig. S3. We can directly apply the on-policy RL algorithms on the resulted rollout to refine the corresponding skill.

S6.2 Dynamical Skill Refinement Mechanism

We first explain why directly refining skills causes skill space collapse which manifests as performance collapse. We then elaborate on the dynamical skill refinement mechanism.

As we have mentioned in Section S4, all the skills are usually embedded into the latent space of the same parametric low-level policy. If we refine a skill, the behaviors of other skills may change stochastically. For a state, the optimal and near-optimal skills usually occupy only a small region of the entire skill space, which means that these skills may have experienced lots of stochastic updates and lost their original behaviors before being sampled by the high-level policy. As shown in Fig. S4, good skill candidates may no longer exist in a specific state. We name this phenomenon skill space collapse. See Appendix SB for the visualization of skill space collapse in specific task.

Intuitively, before a skill is executed in a specific state, its behavior in the state should remain the same as when it was extracted. In addition, since we have theoretically proved the effectiveness of skill refinement, we believe that sufficiently refining a skill in a state can overcome the potential damage to its behavior caused by stochastic changes. We also learn skill refinements into a separate residual policy [17] rather than the low-level policy, which can preserve the extracted behaviors of the skills. The action increment predicted by the residual policy is added to the action predicted by the low-level policy to form the practical action given by the skill. We assign a dynamical weight to action increment, that is, when a skill has been sufficiently refined in a state, the action increment for the state-skill will be given a high weight, otherwise, a low weight. We name this measure dynamical skill refinement (DSR) mechanism.

We use random network distillation (RND) [40] to measure whether the behavior of a skill in a state has been sufficiently refined. It involves two randomly initialized neural networks, namely, fixed target network and variable predictor network. The target network takes a state-skill to an embedding $f : \mathcal{S} \times \mathcal{Z} \to \mathbb{R}^d$. The predictor network $\hat{f} : \mathcal{S} \times \mathcal{Z} \to \mathbb{R}^d$ is trained to minimize the MSE loss $\|\hat{f}((s,z);\xi) - f((s,z))\|^2$. Every time we refine the behavior of skill z in state s, we optimize the prediction error on state-skill (s,z). The prediction errors on the state-skills on which the predictor network has been trained for many times will be obviously lower than those on novel state-skills. We map the prediction error to the weight of action increment. See Appendix SC for the complete algorithm incorporating the dynamical skill refinement mechanism.

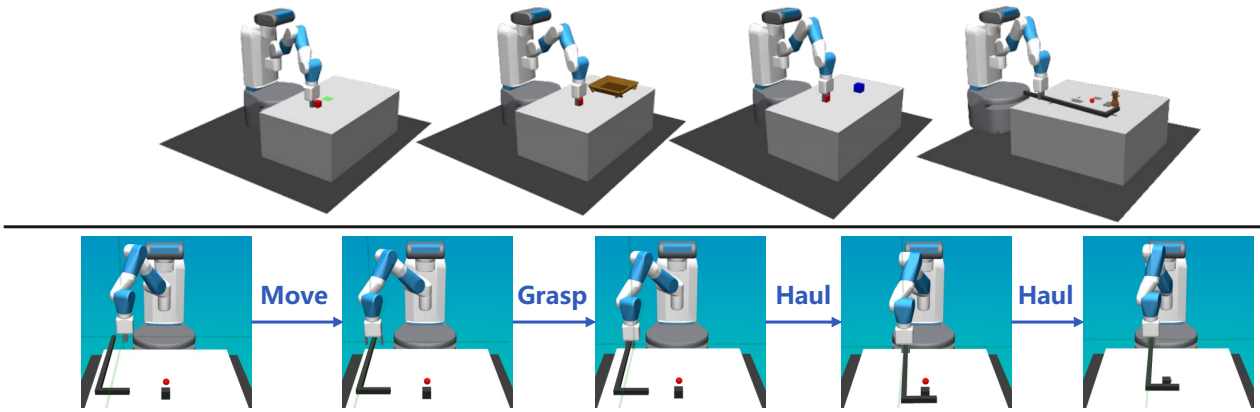


Fig. S5 These tasks are shown in the upper part. (1) **SlipperyPush**: push the cube to the target position. (2) **TableCleanup**: Grab the cube and place it on the tray. (3) **PyramidStack**: Grab the cube and place it on the larger cube. (4) **ComplexHook**: Use the hook to move the object to the target position. The lower part shows the skill sequence to complete ComplexHook.

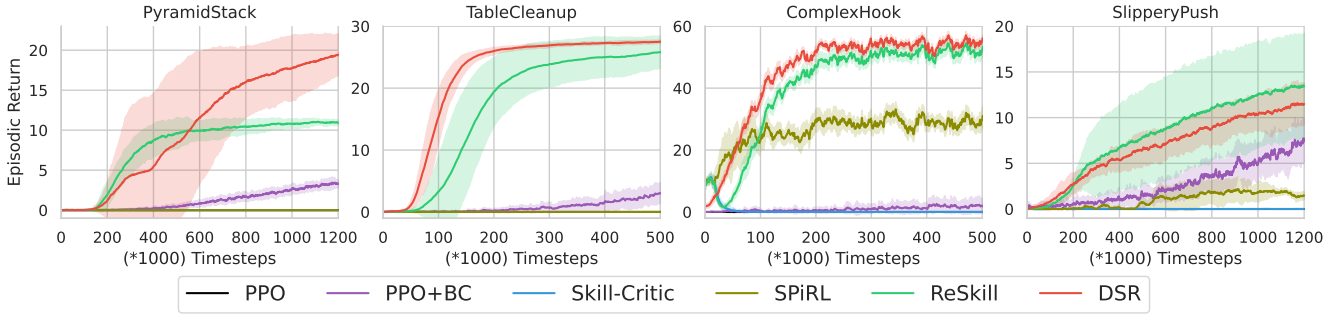


Fig. S6 DSR achieves the highest performance, except for slightly lower performance than ReSkill in SlipperyPush, but DSR shows significantly lower variance. Due to the sub-optimality of skills for downstream tasks, SPiRL can only solve ComplexHook and SlipperyPush, and its performance is limited. Without the warm start phase based on SPiRL, Skill-Critic can only solve ComplexHook in the initial phase and eventually suffers performance collapses in all the tasks. PPO+BC suffers from low sample efficiency, and PPO doesn't even improve performance at all.

S7 Experiments

In the experiments, we first compare the performance of DSR to the SOTA methods in solving sparse-reward tasks. Then, we conduct the ablation analysis on the dynamical skill refinement and the involved hyper-parameters.

We adopt the 4 robotic manipulation tasks proposed in ReSkill [17]. These tasks involve a manipulator arm that can fetch objects, push objects and even move objects using tools. We show these tasks and how the arm solves a task through combining multiple skills in Fig. S5. The demonstration datasets are generated by hand-scripted controller's interacting with the environments. The physically modified versions of these tasks serve as the downstream tasks. These physical modifications necessitate skill refinements. See Appendix SD for the details of experiments.

S7.1 Comparison with SOTA Methods

We compare our method with several SOTA skill-based RL methods. **SPiRL** [5] extracts temporally extended behaviors along with a skill prior from the demonstration dataset and assume these skills are approximately optimal for the downstream task. **Skill-Critic** [15] ignores the temporal abstraction shift and updates the entire hierarchical policy in an off-policy RL manner. **ReSkill** [17] learns skill refinement into the residual policy and updates the entire hierarchical policy in an on-policy RL manner, which circumvents the temporal abstraction shift. We also compare our method with **PPO** [19] and **PPO+BC**. The latter one fine-tunes a policy initialized by behavior cloning through PPO.

For fair comparison, our method (DSR), SPiRL, ReSkill, and Skill-Critic share the same model architecture of VAE. The warm start phase of Skill-Critic with the help of SPiRL is removed. The trick of ReSkill that gradually gives higher weight to the prediction of residual policy with the number of updates is retained. The results are averaged over four random seeds.

The performance curves are presented in Fig. S6. SPiRL can only solve ComplexHook and SlipperyPush with low performance illustrating that the extracted skills are sub-optimal for downstream tasks with modifications on the physical properties.

Only in the initial stage of ComplexHook, the performance of Skill-Critic can be temporarily improved, and the performance collapses in all other times, which shows that the temporal abstraction shift can not be ignored. ReSkill learns the skill refinements into the residual policy, which preserves the original behaviors of the extracted skills to some extent. However, it still suffers significant performance degradation in ComplexHook, which indicates that skill space is still compromised. In contrast, DSR can steadily improve the performance throughout the online learning stage, which verifies the effectiveness of our optimization objective and indicates that skill space collapse is completely avoided. Both PPO and PPO+BC suffer from extremely slow performance improvement. On the contrary, DSR can improve the performance to a high level at a fast speed, which illustrates the necessity of temporal abstraction.

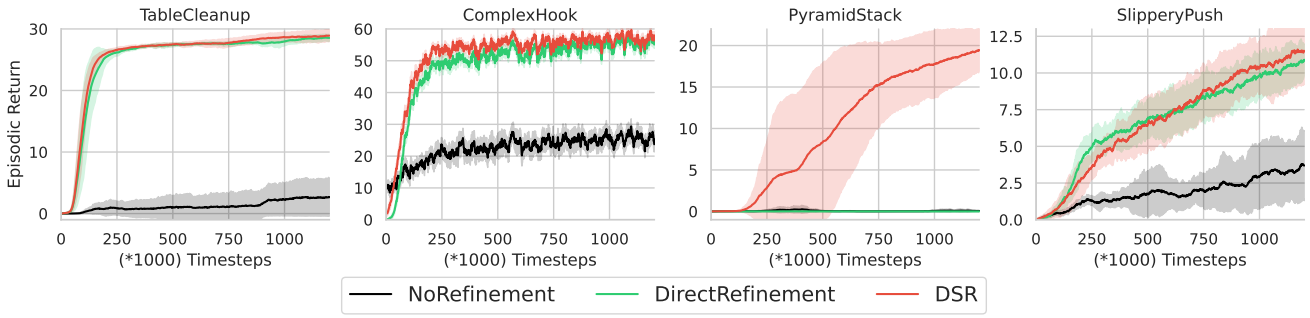


Fig. S7 If skills are kept fixed, the asymptotic performance are limited. The extracted skills are not well initialized for PyramidStack, so the skill space collapse caused by direct skill refinement destroys the low-level policy and the performance is not improved at all. In contrast, when we fine-tune the entire hierarchical policy with dynamical skill refinement mechanism, the performance is improved significantly than fixing the skills. Furthermore, our method can still improve the performance in PyramidStack where direct skill refinement fails, which indicates that the dynamical skill refinement mechanism prevents low-level policy from being destroyed by skill space collapse.

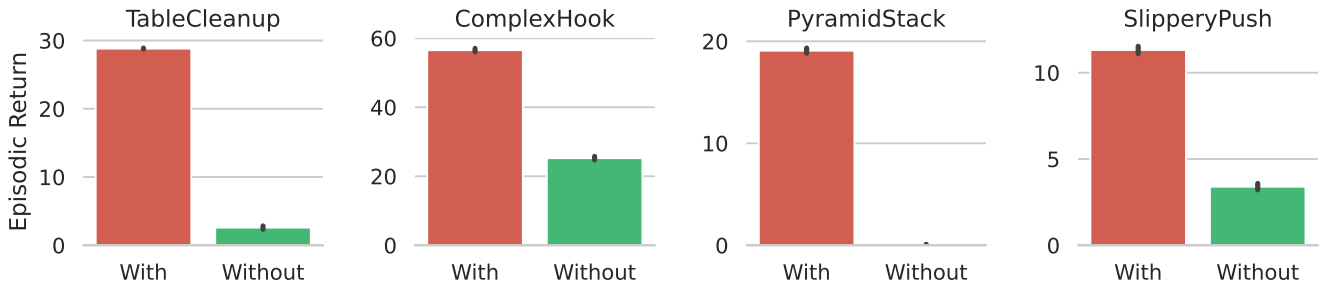
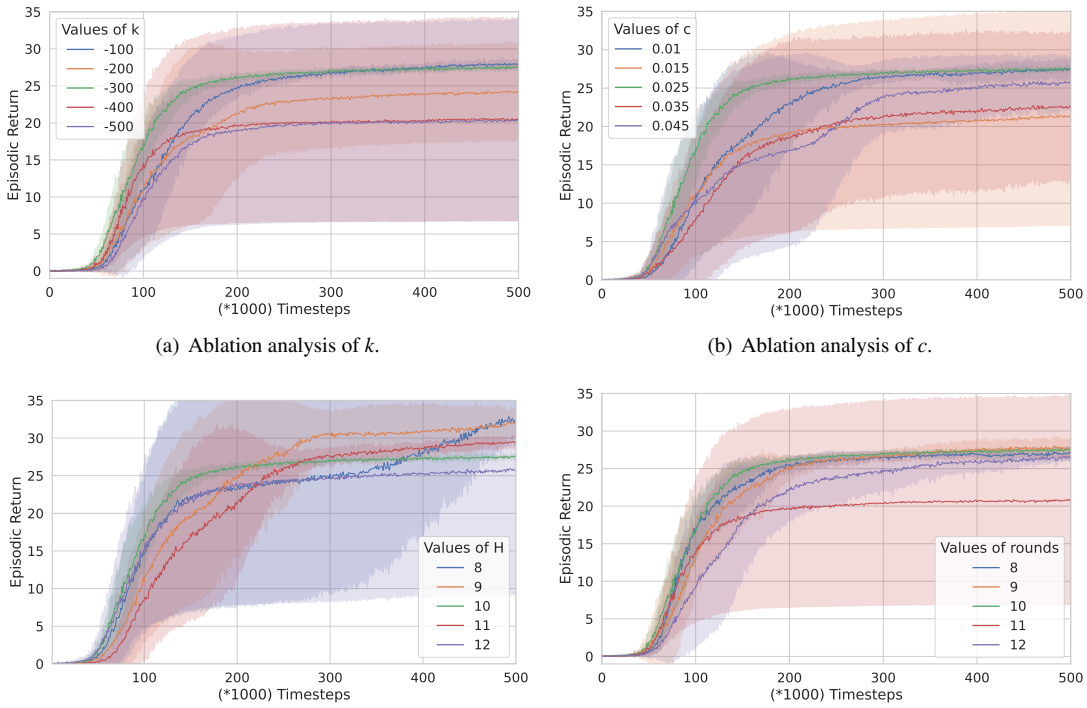


Fig. S8 Under the same high-level policy, skills with refinement lead to higher performance, which obviously benefits from the improved optimality of skills.

S7.2 Ablation Analysis of Dynamical Skill Refinement

In the ablation analysis, we analyze the necessity of dynamical skill refinement, the improvement of skills' optimality, and whether DSR is sensitive to hyper-parameters.

To illustrate the need for skill refinement and the need for the dynamical skill refinement mechanism, we compare the performance of our complete method with the performance of two other cases. In the first case, the skills are not refined (**NoRefinement**). In the second case, the dynamical skill refinement mechanism is removed and the skills are refined directly (**DirectRefinement**). These performance curves are presented in Fig. S7. We find that if the skills are not refined, the asymptotic performance is limited even if the skills are well initialized, which is shown in TableCleanup, ComplexHook and SlipperyPush. If the extracted skills are not well initialized for the downstream task, then the entire hierarchical policy completely fails to solve the task, which is shown in PyramidStack. In contrast, our method achieves obviously higher performance in all the tasks, which illustrates the necessity of refining skills. When the extracted skills are well initialized for the downstream task, the potential skill space collapse may not be enough to destroy the low-level policy, and refining the skills directly can still improve the performance. However, if the extracted skills are not well initialized for the downstream task, the skill space collapse will destroy the low-level policy and the performance can not be improved at all, which is shown in PyramidStack. In contrast, if skills are not well initialized, the dynamical skill refinement mechanism can effectively prevent the fragile low-level policy from being destroyed by skill space collapse. Moreover, in contrast to successful cases of direct



(a) Ablation analysis of k.

(b) Ablation analysis of c.

(c) Ablation analysis of H.

(d) Ablation analysis of rounds.

Fig. S9 We conduct ablation analysis on the three hyper-parameters separately in TableCleanup. Modifying the hyper-parameters in the certain intervals centered in the hand-scripted values results in no significant changes in performance.

skill refinement, we find that though the dynamical skill refinement is a conservative measure, it does not reduce the sample efficiency.

To analyze the improvement in the optimality of skills, we compare the performance achieved by the same sufficiently trained high-level policy when the skills are refined and when they are not. The performance brought about by the skills with refinement (**With**) and the performance brought about by the skills without refinement (**Without**) are shown in Fig. S8. Since the high-level policy is the same, the performance improvement reflects the improvement of skills' optimality.

Finally, we empirically demonstrate the insensitivity of our method to hyper-parameters. The mapping to the weight of action increment from prediction error ($S(x) = \alpha \frac{1}{1+e^{-k(x-c)}}$, see Appendix SD.3 for details, α can be simply set to 1) is mainly determined by two parameters, c and k which are hand-scripted. We determined the values of k and c as -300 and 0.025 for TableCleanup according to the method in the Appendix SD.4. Similarly, the total rounds of variable predictor network's being trained in every epoch is also a hyper-parameter. We set the number of rounds to 10 which is an empirical value. To demonstrate that dynamical skill refinement is insensitive to these hyper-parameters, we respectively adjust them within a certain range and show in Fig. S9 the performance curves of these cases in TableCleanup. Especially, we even conduct ablation analysis on the length of temporal abstraction (H) which is 10 in this paper. The ablation analysis in Fig. S9 demonstrates its robustness to H.

It is evident that adjusting these hyper-parameters within the certain intervals centered in the manually specified values results no significant performance decline, indicating DSR's insensitivity to hyper-parameters.

S8 Conclusion

In this paper, we study the skill-based RL method of updating both high-level policy and skills. We innovatively consider skill-based RL as the proxy task of RL, and theoretically prove that optimizing the performance of the hierarchical policy in TA-MDP is equivalent to optimizing the performance lower bound in original MDP. We devise the optimization objective of skills which is unified with high-level policy and propose to update the entire hierarchical policy in an on-policy RL manner so as to circumvent the temporal abstraction shift. We propose for the first time that direct skill refinement leads to the skill space collapse phenomenon since all the skills are embedded into the latent space of the same parametric policy. To address skill space collapse, we propose the dynamical skill refinement mechanism which can be simply integrated into our algorithm.

Dynamical skill refinement guarantees that the behavior of low-level policy on a given state-skill remains approximate to the extracted behavior before it is sufficiently refined, thus avoiding skill space collapse.

Acknowledgements This work is supported by Shanghai Sailing Program (21YF1421900) and NSFC (62106141).

SA Detailed Theoretical Proof

SA.1 Proof of Lemma 1

$\forall s \in \mathcal{S}$, we denote the value of first predicting a skill z with $\pi^h_{\phi'}$, executing $\pi^l_{\theta'}(\cdot|s,z)$ for H timesteps and then executing π^h_ϕ, π^l_θ by $\tilde{V}^h_{\pi^h_{\phi'},\pi^l_{\theta'},\pi^h_\phi,\pi^l_\theta}(s)$. The following inequality holds, if we follow the update formulae in Eq. S7 and Eq. S8:

$$\forall s \in \mathcal{S}, \tilde{V}^h_{\pi^h_{\phi'},\pi^l_{\theta'},\pi^h_\phi,\pi^l_\theta}(s) \geq V^h_{\pi^h_\phi,\pi^l_\theta}(s). \tag{S13}$$

Proof.

$$\forall s \in \mathcal{S}, \tilde{V}^h_{\pi^h_{\phi'},\pi^l_{\theta'},\pi^h_\phi,\pi^l_\theta}(s) = \tag{S14}$$
$$\mathbb{E}_{z\sim\pi^h_{\phi'}(\cdot|s)}[\mathbb{E}_{a_i\sim\pi^l_{\theta'}(\cdot|s_i,z),s_0=s}[\sum_{i=0}^{H-1} r_i] + \tilde{\gamma}\cdot\mathbb{E}_{s'\sim p_{\pi^l_{\theta'},H}(\cdot|s,z)}[V^h_{\pi^h_\phi,\pi^l_\theta}(s')]].$$

According to the update formula of skills in Eq. S8, we can infer that it is larger than the following equation:

$$\forall s \in \mathcal{S}, \tilde{V}^h_{\pi^h_{\phi'},\pi^l_{\theta'},\pi^h_\phi,\pi^l_\theta}(s) \geq$$
$$\mathbb{E}_{z\sim\pi^h_{\phi'}(\cdot|s)}[\mathbb{E}_{a_i\sim\pi^l_{\theta}(\cdot|s_i,z),s_0=s}[\sum_{i=0}^{H-1} r_i] + \tilde{\gamma}\cdot\mathbb{E}_{s'\sim p_{\pi^l_{\theta},H}(\cdot|s,z)}[V^h_{\pi^h_\phi,\pi^l_\theta}(s')]]$$
$$= \mathbb{E}_{z\sim\pi^h_{\phi'}(\cdot|s)}[Q^h_{\pi^h_\phi,\pi^l_\theta}(s,z)]. \tag{S15}$$

By applying the properties of the high-level policy update formula in Eq. S7, we can further infer that:

$$\forall s \in \mathcal{S}, \mathbb{E}_{z\sim\pi^h_{\phi'}(\cdot|s)}[Q^h_{\pi^h_\phi,\pi^l_\theta}(s,z)] \geq \mathbb{E}_{z\sim\pi^h_{\phi}(\cdot|s)}[Q^h_{\pi^h_\phi,\pi^l_\theta}(s,z)]. \tag{S16}$$

Obviously, it is the value of constantly executing the π^h_ϕ, π^l_θ from state s, namely $V^h_{\pi^h_\phi,\pi^l_\theta}(s)$. In summary, we get the following inequality:

$$\forall s \in \mathcal{S}, \tilde{V}^h_{\pi^h_{\phi'},\pi^l_{\theta'},\pi^h_\phi,\pi^l_\theta}(s) \geq V^h_{\pi^h_\phi,\pi^l_\theta}(s). \tag{S17}$$

□

SA.2 Proof of Theorem 1

If we update π^h_ϕ and π^l_θ following Eq. S7 and Eq. S8 respectively, then $\forall s \in \mathcal{S}, V^h_{\pi^h_{\phi'},\pi^l_{\theta'}}(s) \geq V^h_{\pi^h_\phi,\pi^l_\theta}(s)$.

Proof. We first rewrite $\tilde{V}^h_{\pi^h_{\phi'},\pi^l_{\theta'},\pi^h_\phi,\pi^l_\theta}(s)$ in its expansion:

$$\tilde{V}^h_{\pi^h_{\phi'},\pi^l_{\theta'},\pi^h_\phi,\pi^l_\theta}(s) = \mathbb{E}_{z\sim\pi^h_{\phi'}(\cdot|s)}[\mathbb{E}_{a_i\sim\pi^l_{\theta'}(\cdot|s_i,z),s_0=s}[\sum_{i=0}^{H-1} r_i] +$$
$$\tilde{\gamma}\cdot\mathbb{E}_{s'\sim p_{\pi^l_{\theta'},H}(\cdot|s,z)}[V^h_{\pi^h_\phi,\pi^l_\theta}(s')]]. \tag{S18}$$

We can enlarge $V^h_{\pi^h_\phi,\pi^l_\theta}(s')$ to $\tilde{V}^h_{\pi^h_{\phi'},\pi^l_{\theta'},\pi^h_\phi,\pi^l_\theta}(s')$ according to Lemma 1:

$$\tilde{V}^h_{\pi^h_{\phi'},\pi^l_{\theta'},\pi^h_\phi,\pi^l_\theta}(s) \leq \mathbb{E}_{z\sim\pi^h_{\phi'}(\cdot|s)}[\mathbb{E}_{a_i\sim\pi^l_{\theta'}(\cdot|s_i,z),s_0=s}[\sum_{i=0}^{H-1} r_i] +$$
$$\tilde{\gamma}\cdot\mathbb{E}_{s'\sim p_{\pi^l_{\theta'},H}(\cdot|s,z)}[\tilde{V}^h_{\pi^h_{\phi'},\pi^l_{\theta'},\pi^h_\phi,\pi^l_\theta}(s')]]. \tag{S19}$$

We can expand the $\tilde{V}^h_{\pi^h_{\phi'},\pi^l_{\theta'},\pi^h_{\phi},\pi^l_{\theta}}(s')$ and get:

$$= \mathbb{E}_{z\sim\pi^h_{\phi'}(\cdot|s)}[\mathbb{E}_{a_i\sim\pi^l_{\theta'}(\cdot|s_i,z),s_0=s}[\sum_{i=0}^{H-1} r_i]+ \tag{S20}$$

$$\tilde{\gamma}\cdot\mathbb{E}_{s'\sim p_{\pi^l_{\theta'},H}(\cdot|s,z)}[\mathbb{E}_{z'\sim\pi^h_{\phi'}(\cdot|s')}[\mathbb{E}_{a_j\sim\pi^l_{\theta'}(\cdot|s_j,z'),s_H=s'}[\sum_{j=H}^{2H-1} r_j]+$$

$$\tilde{\gamma}\cdot\mathbb{E}_{s''\sim p_{\pi^l_{\theta'},H}(\cdot|s',z')}[V^h_{\pi^h_{\phi},\pi^l_{\theta}}(s'')]]]].$$

If we expand $\tilde{V}^h_{\pi^h_{\phi'},\pi^l_{\theta'},\pi^h_{\phi},\pi^l_{\theta}}(\cdot)$ and enlarge $V^h_{\pi^h_{\phi},\pi^l_{\theta}}(\cdot)$ to $\tilde{V}^h_{\pi^h_{\phi'},\pi^l_{\theta'},\pi^h_{\phi},\pi^l_{\theta}}(\cdot)$ infinitely and repeatedly, we will obviously get:

$$\tilde{V}^h_{\pi^h_{\phi'},\pi^l_{\theta'},\pi^h_{\phi},\pi^l_{\theta}}(s) \le ... \le ... = V^h_{\pi^h_{\phi'},\pi^l_{\theta'}}(s). \tag{S21}$$

Taking Lemma 1 into account, we get the following inequality:

$$\forall s\in\mathcal{S}, V^h_{\pi^h_{\phi},\pi^l_{\theta}}(s) \le \tilde{V}^h_{\pi^h_{\phi'},\pi^l_{\theta'},\pi^h_{\phi},\pi^l_{\theta}}(s) \le V^h_{\pi^h_{\phi'},\pi^l_{\theta'}}(s). \tag{S22}$$

In summary, the following inequality holds:

$$\forall s\in\mathcal{S}, V^h_{\pi^h_{\phi'},\pi^l_{\theta'}}(s) \ge V^h_{\pi^h_{\phi},\pi^l_{\theta}}(s). \tag{S23}$$

□

SA.3 Proof of Theorem 2

With the policy update formulae in Eq. S7 and Eq. S8, the state value function $V^h_{\pi^h_{\phi_k},\pi^l_{\theta_k}}(s)$ will finally converge. $\pi^h_{\phi_k},\pi^l_{\theta_k}$ are the k-th version of high-level policy and low-level policy respectively.

Proof. Obviously, the H-timestep rewards have an upper bound $H\cdot r_{\max}$ where $r_{\max}$ is the maximum reward given by the original MDP. Therefore, there is a value upper bound in the TA-MDP:

$$\frac{H\cdot r_{\max}}{1-\tilde{\gamma}}. \tag{S24}$$

Based on Theorem 1, we can know that $V^h_{\pi^h_{\phi_k},\pi^l_{\theta_k}}(s)$ increases monotonically. In summary, $V^h_{\pi^h_{\phi_k},\pi^l_{\theta_k}}(s)$ converges as $k\to\infty$. □

SA.4 Proof of Theorem 3

If the MDP $(\mathcal{S},\mathcal{A},p,r,\gamma)$ and the TA-MDP $(\mathcal{S},\mathcal{Z},p_{\pi^l,H},\tilde{r},\tilde{\gamma})$ satisfy that $\tilde{\gamma}=\gamma^H$, then $\forall s\in\mathcal{S}, \tilde{\gamma}\cdot V^h_{\pi^h,\pi^l}(s) \le V_{\pi^h,\pi^l}(s)$. It means that optimizing $V^h_{\pi^h,\pi^l}(s)$ is equivalent to optimizing a lower bound of $V_{\pi^h,\pi^l}(s)$.

Proof. We can expand $\tilde{\gamma}\cdot V^h_{\pi^h,\pi^l}(s)$ with Eq. S12 as follows:

$$\tilde{\gamma}\cdot V^h_{\pi^h,\pi^l}(s)$$

$$= \tilde{\gamma}\cdot\int\sum_{\triangle t=0}^{\infty}\rho_{\triangle t}(s',a'|s,\pi^h,\pi^l)\tilde{\gamma}^{\lfloor\triangle t/H\rfloor}\cdot r(s',a')da'ds' \tag{S25}$$

$$= \int\sum_{\triangle t=0}^{\infty}\rho_{\triangle t}(s',a'|s,\pi^h,\pi^l)\tilde{\gamma}^{\lfloor\triangle t/H\rfloor+1}\cdot r(s',a')da'ds'. \tag{S26}$$

Because r is always non-negative and the discount factor $\tilde{\gamma}\in(0,1]$, we know $\tilde{\gamma}\cdot V^h_{\pi^h,\pi^l}(s)$ satisfies that:

$$\tilde{\gamma}\cdot V^h_{\pi^h,\pi^l}(s) \le \int\sum_{\triangle t=0}^{\infty}\rho_{\triangle t}(s',a'|s,\pi^h,\pi^l)\tilde{\gamma}^{\triangle t/H}\cdot r(s',a')da'ds'. \tag{S27}$$

We can replace $\tilde{\gamma}$ with γ^H and get:

$$= \int \sum_{\triangle t=0}^{\infty} \rho_{\triangle t}(s', a'|s, \pi^h, \pi^l)(\gamma^H)^{\triangle t/H} \cdot r(s', a')da'ds' \tag{S28}$$

$$= \int \sum_{\triangle t=0}^{\infty} \rho_{\triangle t}(s', a'|s, \pi^h, \pi^l)\gamma^{\triangle t} \cdot r(s', a')da'ds' \tag{S29}$$

$$= V_{\pi^h,\pi^l}(s). \tag{S30}$$

In summary, $\forall s \in \mathcal{S}, \tilde{\gamma} \cdot V^h_{\pi^h,\pi^l}(s) \leq V_{\pi^h,\pi^l}(s)$, which means that optimizing $V^h_{\pi^h,\pi^l}(s)$ is equivalent to optimizing a lower bound of $V_{\pi^h,\pi^l}(s)$. □

SB Visualization of Skill Space Collapse

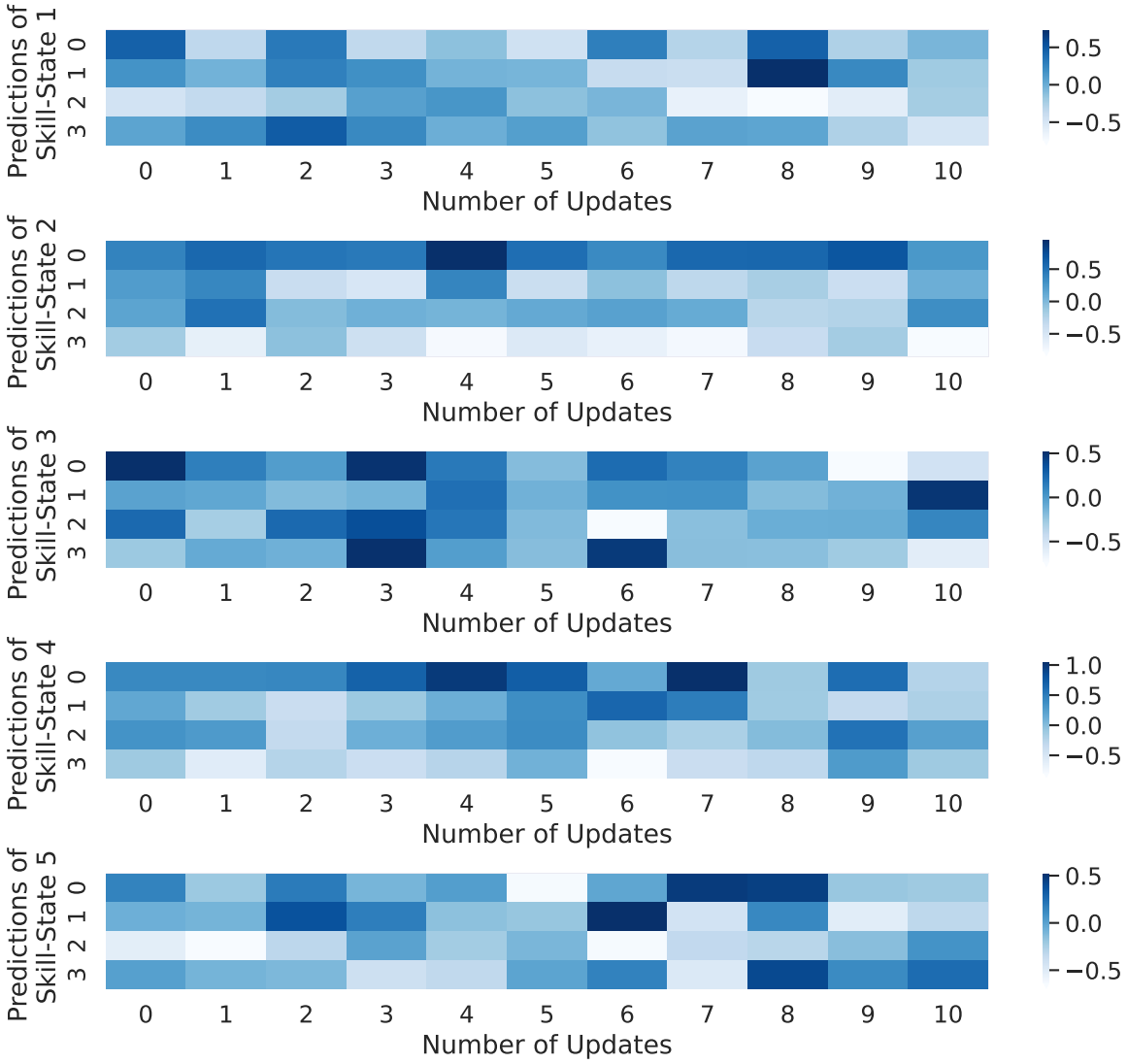


Fig. S10 Each sub-figure represents the change process of actions predicted by the low-level policy for a state-skill. Each column of the sub-figure represents a predicted action, and the four rows in this column represent the four components of the action. The i-th column represents the predicted action after i updates of the hierarchical policy. The predicted actions after two updates are significantly different from the initial predicted actions.

To visually show the process in which skill space collapse occurs, we visualize the change of the predicted actions given by the low-level policy for 5 state-skills that are not used for updating as the whole hierarchical policy is updated. This process of change takes place in the PyramidStack task adopted in Section S7 and is shown in Fig. S10. We find that after 1 update, the predicted actions still remain highly similar to the initial predicted actions. However, after 2 and more updates, the predicted actions and the initial predicted actions are obviously different, although the low-level policy is never updated with respect to these state-skills.

SC Algorithm of Updating High-level Policy and Skills Simultaneously

Algorithm S1 DSR: Dynamical Skill Refinement in Skill-based RL

Require: Pretrained high-level policy π^h_{off}, extracted low-level policy π^l_{off}, length of temporal abstraction H
Ensure: High-level policy π^h_ϕ, residual low-level policy π^l_θ, target network $\hat{f}$, variable network f_ξ

Initialize $\pi^h_\phi = \pi^h_{\text{off}}$ and create $\pi^l_\theta, \hat{f}, f_\xi$
for each epoch **do**
  Create high-level buffer $\beta^h = \emptyset$ and low-level buffer $\beta^l = \emptyset$
  for each episode **do**
    Initialize trajectory queue $tq = []$
    for timestep $t = 0, 1, 2, ...$ **do**
      if $t \mod H = 0$ **then**
        Sample a skill $z \sim \pi^h_\phi(\cdot|s_t)$
      end if
      Decode an action $a_t \sim \pi^l_{\text{off}}(\cdot|s_t, z)$ and an action increment $\hat{a}_t \sim \pi^l_\theta(\cdot|s_t, z)$
      Map prediction error $\|\hat{f}((s_t, z); \xi) - f((s, z))\|_2$ to a weight ω_t
      Add action increment to action $a_t \leftarrow a_t + \omega_t \cdot \hat{a}_t$
      Execute a_t and get r_t, s_{t+1}
      Append $(s_t, z, \hat{a}_t, r_t, s_{t+1})$ to tq
    end for
    Extract high-level trajectory from tq and store it:
    $\beta^h \leftarrow \beta^h \cup \{(..., (s_{\hat{t}\cdot H}, z_{\hat{t}\cdot H}, \hat{r}_{\hat{t}\cdot H}, s_{(\hat{t}+1)\cdot H}), ...)\}$
    Divide tq into H-timestep rollouts and store them:
    $\beta^l \leftarrow \beta^l \cup \{..., ((s_{\hat{t}\cdot H}, z_{\hat{t}\cdot H}, \hat{a}_{\hat{t}\cdot H}, r_{\hat{t}\cdot H}, s_{\hat{t}\cdot H+1}), ...,$
    $(s_{\hat{t}\cdot H+H-1}, z_{\hat{t}\cdot H}, \hat{a}_{\hat{t}\cdot H+H-1}, r_{\hat{t}\cdot H+H-1}, s_{\hat{t}\cdot H+H})), ...\}$
    where each $r_{\hat{t}\cdot H+H-1}$ has been processed as Fig. S3
  end for
  Apply PPO on β^h to update π^h_ϕ
  Apply PPO on β^l to update π^l_θ
  Minimize $\|\hat{f}(\cdot; \xi) - f(\cdot)\|_2$ with all (s, z) in β^l to update ξ
end for
return $\pi^h_\phi, \pi^l_\theta, \hat{f}, f_\xi$

In the online learning stage, the high-level policy and the skills interact with the environment to generate the trajectories which can be converted to the trajectories in the TA-MDP and divided into the constructed rollouts for refining the skills. We can directly use PPO [19] to finetune the high-level policy. With the constructed rollouts, the skills can also be directly refined through PPO. Notably, when the behavior of a skill z in a state s is refined, the predictor network's prediction error $\|\hat{f}((s, z); \xi) - f((s, z))\|_2$ regarding the state-skill (s, z) should be accordingly optimized.

During the interaction between the hierarchical policy and the environment, when a skill z predicts an action in the current state s, the residual policy predicts the action increment of the skill in the state. We convert the prediction error of the predictor network in this state-skill (s, z) into a weight in the interval $[0, 1]$, assign this weight to the action increment, and add the weighted action increment to the predicted action to get the practical action. This conversion can be implemented by a scaled and shifted Sigmoid function.

The complete algorithm that incorporates dynamical skill refinement is detailedly illustrated in Algorithm S1. Obviously, the high-level policy and skills are updated only at the end of each epoch. During the epoch, all the skills remain fixed, which

means that the transition dynamics of the TA-MDP are stationary. Due to the natural circumvention of the temporal abstraction shift, we can directly use the PPO algorithm to update both the high-level policy and skills. The dynamical skill refinement mechanism is easily integrated into our algorithm. Its predictor network is updated alongside the skills, and the only way it influences the action is by assigning a dynamical weight to the action increment.

SD Details of Experiments

SD.1 Discount the Inner-skill Rewards

We find that the values of all the inner-skill state-actions are the same when the reward function is sparse, which is shown in Fig. S3. However, in sparse-reward tasks, we want to solve the task as soon as possible; therefore, we propose to optimize the expected sum of the single-timestep rewards which are discounted by $\gamma = \tilde{\gamma}^{\frac{1}{H}}$ after every timestep rather than being discounted by $\tilde{\gamma}$ after every H timesteps. We prove theoretically in Corollary 1 that such a skill refinement is equivalent to optimizing the lower bound of the performance in TA-MDP, which means that it is still unified with the high-level policy.

Corollary 1. If the original MDP $(\mathcal{S}, \mathcal{A}, p, r, \gamma)$ and the TA-MDP $(\mathcal{S}, \mathcal{Z}, p_{\pi^l,H}, \tilde{r}, \tilde{\gamma})$ satisfy that $\tilde{\gamma} = \gamma^H$, then $\forall s \in \mathcal{S}, V_{\pi^h,\pi^l}(s) \leq V^h_{\pi^h,\pi^l}(s)$.

Proof. We can expand $V_{\pi^h,\pi^l}(s)$ with Eq. S11 as follows:

$$V_{\pi^h,\pi^l}(s) = \int \sum_{\triangle t=0}^{\infty} \rho_{\triangle t}(s', a'|s, \pi^h, \pi^l)\gamma^{\triangle t} \cdot r(s', a')da'ds'. \tag{S31}$$

Then, we rewrite it in the equivalent form:

$$= \int \sum_{\triangle t=0}^{\infty} \rho_{\triangle t}(s', a'|s, \pi^h, \pi^l)(\gamma^H)^{\triangle t/H} \cdot r(s', a')da'ds' \tag{S32}$$

$$= \int \sum_{\triangle t=0}^{\infty} \rho_{\triangle t}(s', a'|s, \pi^h, \pi^l)\tilde{\gamma}^{\triangle t/H} \cdot r(s', a')da'ds'. \tag{S33}$$

We enlarge it and get:

$$V_{\pi^h,\pi^l}(s) \leq \int \sum_{\triangle t=0}^{\infty} \rho_{\triangle t}(s', a'|s, \pi^h, \pi^l)\tilde{\gamma}^{\lfloor \triangle t/H \rfloor} \cdot r(s', a')da'ds' \tag{S34}$$

$$= V^h_{\pi^h,\pi^l}(s). \tag{S35}$$

In summary, if we refine the skills with respect to the expected sum of the single-timestep rewards discounted by $\tilde{\gamma}^{\frac{1}{H}}$, we actually optimize a lower bound of the state-value in the TA-MDP. □

Intuitively, we can consider such a skill refinement as optimizing the low-level policy π^l in an MDP $(\hat{\mathcal{S}}, \hat{\mathcal{A}}, \hat{p}_{\pi^h}, \hat{r}, \hat{\gamma})$. The state space $\hat{\mathcal{S}} = \mathcal{S} \times \mathcal{Z}$ is Cartesian product of the state space $\mathcal{S}$ of the original MDP and the skill space $\mathcal{Z}$. The action space $\hat{\mathcal{A}} = \mathcal{A}$ is equal to the original action space. The transition dynamics $\hat{p}_{\pi^h}$ is determined by the original transition dynamics p and the high-level policy π^h. The reward function $\hat{r} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ maps the original state component and the action to the reward value which the original reward function will also give. The discount factor $\hat{\gamma} = \tilde{\gamma}^{\frac{1}{H}}$. The trajectory generated by the interaction of the entire hierarchical policy with the environment can be viewed as the rollout of the low-level policy. From this perspective, we can directly use the PPO [19] algorithm to optimize the low-level policy. We find in experiments that this practical implementation of skill refinement works well.

SD.2 Details of Datasets and Downstream Tasks

We reuse the datasets from ReSkill [17], which were collected by hand-scripted controllers. The fetch_block_40000 dataset consists of 40,000 trajectories collected from three tasks: TableCleanup, SlipperyPush and PyramidStack. We extract the skills from it for the physically modified versions of these tasks which serve as downstream tasks. The fetch_hook_40000 dataset consists of 40,000 trajectories collected from ComplexHook. We extract the skills from it for the physically modified version of ComplexHook which serves as the downstream task.

When these tasks are used as downstream tasks, their physical properties are modified appropriately in order to make skill refinement more necessary. In SlipperyPush, the friction of the table surface is reduced compared to the transitions in the dataset. In TableCleanup, the tray is actually added and the agent has to overcome the edges of the tray which are the obstacles. In PyramidStack, the gripper can reach higher heights. In ComplexHook, the objects are drawn randomly from a library of unseen objects and the table surface is scattered rigid obstacles. The lengths of episodes in TableCleanup, PyramidStack, SlipperyPush and ComplexHook are 50, 50, 100 and 100 respectively. These details remain consistent with ReSkill.

SD.3 Map Prediction Error to the Weight of Action Increment

Downstream Task	Hyper-parameter	Value
TableCleanup	α	1
TableCleanup	k	-300
TableCleanup	c	0.025
SlipperyPush	α	1
SlipperyPush	k	-300
SlipperyPush	c	0.025
PyramidStack	α	0.6
PyramidStack	k	-10
PyramidStack	c	0.01
ComplexHook	α	1
ComplexHook	k	-20
ComplexHook	c	0.04

Table S1 Hyper-parameters of mapping prediction error to weight of action increment.

We point out that the prediction error given by distillation networks can be mapped to the weight of the action increment by the following scaled and shifted Sigmoid function.

$$S(x) = \alpha \frac{1}{1 + e^{-k(x-c)}}. \tag{S36}$$

α, k, c can be seen as the hand-drafted task-specific hyper-parameters. The values of these hyper-parameters in different downstream tasks are shown in Table S1. We can collect very few transitions generated by agent's interacting with downstream tasks and visualize the prediction error curves to determine appropriate values of these hyper-parameters. The architectures of the prediction and target networks are the same and illustrated in Table S2.

Properties	Value
Hidden sizes	[64, 64]
Activation function	Tanh
Output activation function	Identity
Optimizer	Adam
Learning rate	$3e-4$

Table S2 Architecture of the prediction and the target networks.

SD.4 Determine α, k, c with a few of online interactions

We use the TableCleanup task as an example to show how to determine the hyper-parameters α, k, c used to map the prediction error to the weight of the action increment. We use a few of online interactions to select the appropriate values of α, k, c. We use only 50 epochs to tentatively train the hierarchical policy and variable prediction network and record the prediction errors. We visualize these prediction errors in Fig. S11. We find that the prediction errors concentrate below 0.025 as the training progresses. Before that, the prediction errors hover in $(0.025, 0.04]$. When the prediction errors are above 0.04, they show a trend of divergence. It is natural to consider the state-skill with a prediction error less than 0.025 as a sufficiently refined state-skill. This observation motivates us to set c to 0.025, which means that when the prediction error is around 0.025, the mapped weight increases rapidly. When the prediction error approaches 0, we expect the weight to approach 1. With this goal, we can simply set α equal to 1 and just adjust k. Since the adjustment process is done on the collected data, it involves no online interaction costs. The mapping from prediction error to weight obtained by adjusting k is presented in Fig. S12.

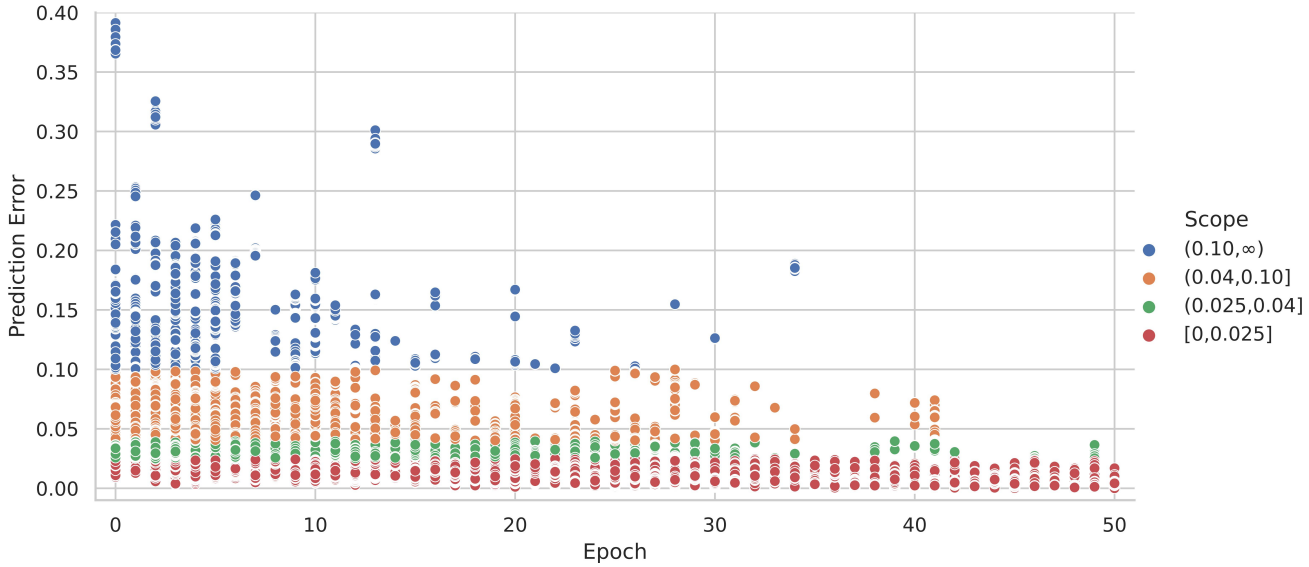


Fig. S11 The prediction errors are given different colors according to the scopes to which they belong.

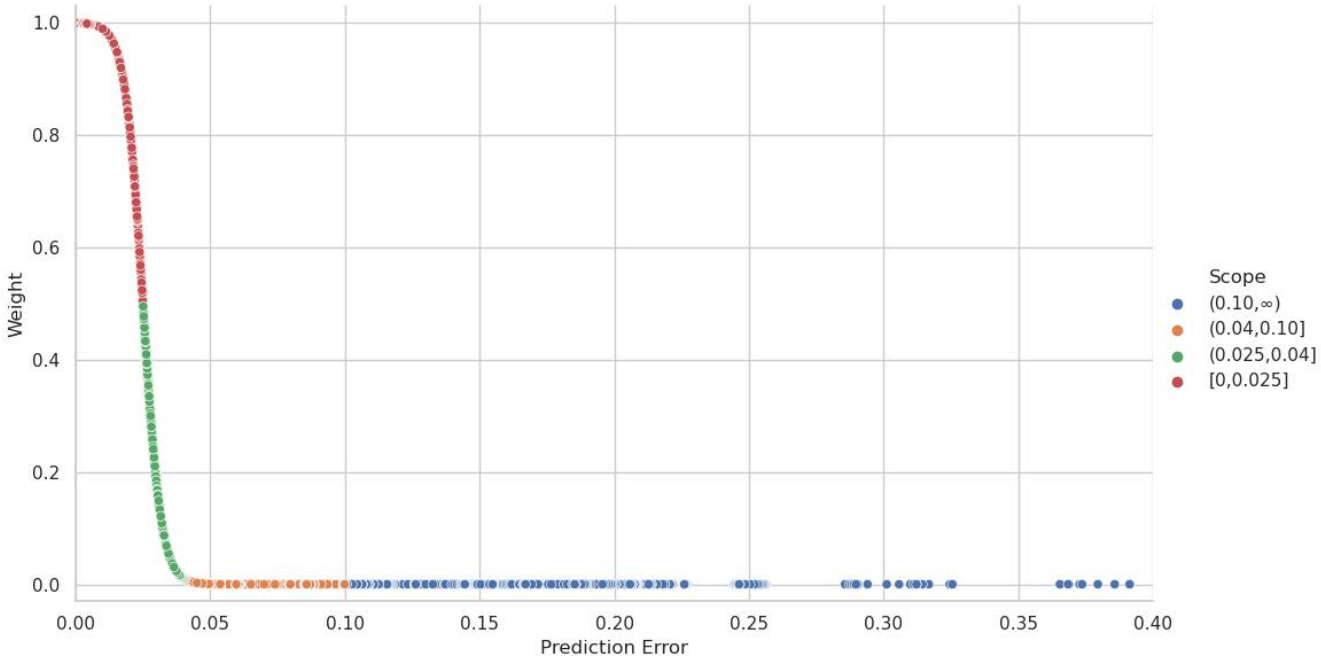


Fig. S12 After adjusting, by setting $\alpha = 1$ and $c = 0.025$, we find that $k = -300$ works as expected.

References

1. Gupta A, Kumar V, Lynch C, Levine S, Hausman K. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. arXiv preprint arXiv:1910.11956, 2019
2. Chane-Sane E, Schmid C, Laptev I. Goal-conditioned reinforcement learning with imagined subgoals. In: International conference on machine learning. 2021, 1430–1440
3. Kipf T, Li Y, Dai H, Zambaldi V, Sanchez-Gonzalez A, Grefenstette E, Kohli P, Battaglia P. Compile: Compositional imitation learning and execution. In: International Conference on Machine Learning. 2019, 3418–3428
4. Nair A, Gupta A, Dalal M, Levine S. Awac: Accelerating online reinforcement learning with offline datasets. arXiv preprint arXiv:2006.09359, 2020
5. Pertsch K, Lee Y, Lim J. Accelerating reinforcement learning with learned skill priors. In: Conference on robot learning. 2021, 188–204
6. Shi L X, Lim J J, Lee Y. Skill-based model-based reinforcement learning. arXiv preprint arXiv:2207.07560, 2022

7. Huang T, Chen K, Li B, Liu Y H, Dou Q. Guided reinforcement learning with efficient exploration for task automation of surgical robot. In: 2023 IEEE International Conference on Robotics and Automation (ICRA). 2023, 4640–4647
8. Pertsch K, Lee Y, Wu Y, Lim J J. Guided reinforcement learning with learned skills. In: Conference on Robot Learning. 2022, 729–739
9. Rajeswaran A, Kumar V, Gupta A, Vezzani G, Schulman J, Todorov E, Levine S. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. arXiv preprint arXiv:1709.10087, 2017
10. Daoudi P, Robu B, Prieur C, Dos Santos L, Barlier M. Enhancing reinforcement learning agents with local guides. In: International Conference on Autonomous Agents and Multiagent Systems. 2023
11. Zhang H, Xu W, Yu H. Policy expansion for bridging offline-to-online reinforcement learning. arXiv preprint arXiv:2302.00935, 2023
12. Wen X, Yu X, Yang R, Bai C, Wang Z. Towards robust offline-to-online reinforcement learning via uncertainty and smoothness. arXiv preprint arXiv:2309.16973, 2023
13. Guo S, Sun Y, Hu J, Huang S, Chen H, Piao H, Sun L, Chang Y. A simple unified uncertainty-guided framework for offline-to-online reinforcement learning. arXiv preprint arXiv:2306.07541, 2023
14. Fu J, Kumar A, Nachum O, Tucker G, Levine S. D4rl: Datasets for deep data-driven reinforcement learning. arXiv preprint arXiv:2004.07219, 2020
15. Hao C, Weaver C, Tang C, Kawamoto K, Tomizuka M, Zhan W. Skill-critic: Refining learned skills for hierarchical reinforcement learning. IEEE Robotics and Automation Letters, 2024
16. Haarnoja T, Zhou A, Abbeel P, Levine S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: International conference on machine learning. 2018, 1861–1870
17. Rana K, Xu M, Tidd B, Milford M, Sünderhauf N. Residual skill policies: Learning an adaptable skill-based action space for reinforcement learning for robotics. In: Conference on Robot Learning. 2023, 2095–2104
18. Schulman J, Levine S, Abbeel P, Jordan M, Moritz P. Trust region policy optimization. In: International conference on machine learning. 2015, 1889–1897
19. Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347, 2017
20. Pateria S, Subagdja B, Tan A h, Quek C. Hierarchical reinforcement learning: A comprehensive survey. ACM Computing Surveys (CSUR), 2021, 54(5): 1–35
21. Li A C, Florensa C, Clavera I, Abbeel P. Sub-policy adaptation for hierarchical reinforcement learning. arXiv preprint arXiv:1906.05862, 2019
22. Li J, Tang C, Tomizuka M, Zhan W. Hierarchical planning through goal-conditioned offline reinforcement learning. IEEE Robotics and Automation Letters, 2022, 7(4): 10216–10223
23. Zhang S, Whiteson S. Dac: The double actor-critic architecture for learning options. Advances in Neural Information Processing Systems, 2019, 32
24. Yang Y, Hu H, Li W, Li S, Yang J, Zhao Q, Zhang C. Flow to control: Offline reinforcement learning with lossless primitive discovery. In: Proceedings of the AAAI Conference on Artificial Intelligence. 2023, 10843–10851
25. Ajay A, Kumar A, Agrawal P, Levine S, Nachum O. Opal: Offline primitive discovery for accelerating offline reinforcement learning. arXiv preprint arXiv:2010.13611, 2020
26. Daniel C, Neumann G, Kroemer O, Peters J. Hierarchical relative entropy policy search. Journal of Machine Learning Research, 2016, 17(93): 1–50
27. Huang Z, Liang L, Ling Z, Li X, Gan C, Su H. Reparameterized policy learning for multimodal trajectory optimization. In: International Conference on Machine Learning. 2023, 13957–13975
28. Bacon P L, Harb J, Precup D. The option-critic architecture. In: Proceedings of the AAAI conference on artificial intelligence. 2017
29. Haarnoja T, Hartikainen K, Abbeel P, Levine S. Latent space policies for hierarchical reinforcement learning. In: International Conference on Machine Learning. 2018, 1851–1860
30. Levy A, Konidaris G, Platt R, Saenko K. Learning multi-level hierarchies with hindsight. In: Proceedings of International Conference on Learning Representations. 2019
31. Shankar T, Tulsiani S, Pinto L, Gupta A. Discovering motor programs by recomposing demonstrations. In: International Conference on Learning Representations. 2019
32. Krishnan S, Fox R, Stoica I, Goldberg K. Ddco: Discovery of deep continuous options for robot learning from demonstrations. In: Conference on robot learning. 2017, 418–437
33. Nachum O, Gu S, Lee H, Levine S. Near-optimal representation learning for hierarchical reinforcement learning. arXiv preprint arXiv:1810.01257, 2018
34. Hausman K, Springenberg J T, Wang Z, Heess N, Riedmiller M. Learning an embedding space for transferable robot skills. In: International Conference on Learning Representations. 2018
35. Celik O, Zhou D, Li G, Becker P, Neumann G. Specializing versatile skill libraries using local mixture of experts. In: Conference on Robot Learning. 2022, 1423–1433
36. Kingma D P, Welling M. Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114, 2013
37. Walke H R, Yang J H, Yu A, Kumar A, Orbik J, Singh A, Levine S. Don't start from scratch: Leveraging prior data to automate robotic reinforcement

learning. In: Conference on Robot Learning. 2023, 1652–1662

38. Xu M, Veloso M, Song S. Aspire: Adaptive skill priors for reinforcement learning. Advances in Neural Information Processing Systems, 2022, 35: 38600–38613
39. Nam T, Sun S H, Pertsch K, Hwang S J, Lim J J. Skill-based meta-reinforcement learning. arXiv preprint arXiv:2204.11828, 2022
40. Burda Y, Edwards H, Storkey A, Klimov O. Exploration by random network distillation. In: Seventh International Conference on Learning Representations. 2019, 1–17