PIPER: Primitive-Informed Preference-based Hierarchical Reinforcement Learning via Hindsight Relabeling

Utsav Singh
CSE Deptt.
IIT Kanpur, India
utsavz@iitk.ac.in

Wesley A. Suttle
U.S. Army Research Laboratory
Adelphi, MD, USA
wesley.a.suttle.ctr@army.mil

Brian M. Sadler
University of Texas
Austin, Texas, USA

Vinay P Namboodiri
CS Deptt.
University of Bath, Bath, UK
vpn22@bath.ac.uk

Amrit Singh Bedi
CS Deptt., University of Central Florida
Orlando, Florida, USA
amritbedi@ucf.edu

Abstract

In this work, we introduce PIPER: Primitive-Informed Preference-based Hierarchical reinforcement learning via Hindsight Relabeling, a novel approach that leverages preference-based learning to learn a reward model, and subsequently uses this reward model to relabel higher-level replay buffers. Since this reward is unaffected by lower primitive behavior, our relabeling-based approach is able to mitigate non-stationarity, which is common in existing hierarchical approaches, and demonstrates impressive performance across a range of challenging sparse-reward tasks. Since obtaining human feedback is typically impractical, we propose to replace the human-in-the-loop approach with our primitive-in-the-loop approach, which generates feedback using sparse rewards provided by the environment. Moreover, in order to prevent infeasible subgoal prediction and avoid degenerate solutions, we propose primitive-informed regularization that conditions higher-level policies to generate feasible subgoals. We perform extensive experiments to show that PIPER mitigates non-stationarity in hierarchical reinforcement learning and achieves greater than 50% success rates in challenging, sparse-reward robotic environments, where most other baselines fail to achieve any significant progress.

1 Introduction

Deep reinforcement learning (RL) has propelled significant advances in sequential decision-making tasks, where an agent learns complex behaviors through trial and error. Examples of such tasks include playing Atari games [26], mastering the game of Go [36], and performing complicated robotic manipulation tasks [35, 16, 11, 23]. However, the success of RL-based approaches is impeded by ineffective exploration and long-term credit assignment, especially in sparse-reward scenarios [30].

Off-policy hierarchical reinforcement learning (HRL) [37, 9, 38, 18, 14] approaches hold the promise of improved sample efficiency due to temporal abstraction and improved performance due to more effective exploration [29]. In goal-conditioned HRL [9, 38], higher-level policies predict subgoals for the lower-level policies, while the lower-level policies try to achieve those subgoals by executing primitive actions directly in the environment. Despite their advantages, serious challenges remain for such approaches. In this paper, we focus on two important issues: the destabilizing effect of

38th Workshop on Aligning Reinforcement Learning Experimentalists and Theorists (ARLET 2024).

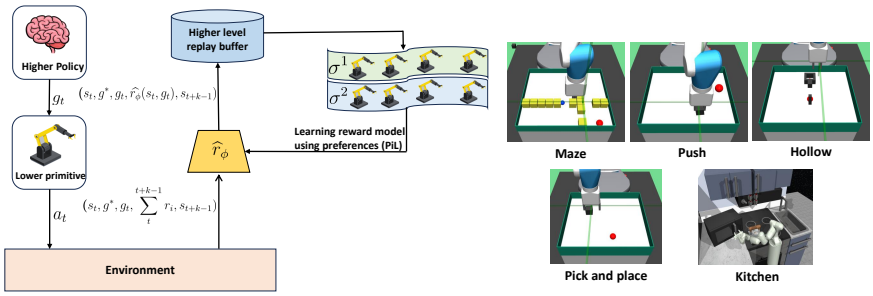


Figure 1: **PIPER Overview** This figure shows the overview of PIPER (left). The higher level policy predicts subgoals g_t for the lower primitive, which executes actions a_t on the environment. We propose to learn a preference-based reward model $\hat{r}_\phi$ using our PiL feedback on higher level trajectories sampled from higher level replay buffer, and subsequently use $\hat{r}_\phi$ to relabel the replay buffer transitions, thereby mitigating non-stationarity in HRL. On the right, we depict the training environments: (i) maze navigation environment, (ii) pick and place environment, (iii) push environment, (iv) hollow environment, and (v) franka kitchen environment.

lower-level reward non-stationarity on off-policy HRL [24, 28], and performance degradation due to infeasible subgoal generation by higher-level policies [5].

Recent work in preference-based learning [6, 15, 21] uses human feedback to learn a reward model r_ϕ, and subsequently apply RL to solve the task using r_ϕ. Impressively, the performance of such methods is comparable to RL policies trained with access to ground truth rewards. This raises the following question: *can advances in preference-based learning be used to simultaneously address the twin issues of reward non-stationarity and infeasible subgoal generation in HRL?*

In this work, we provide an affirmative answer to the above question by proposing PIPER: **P**rimitive **I**nformed **P**r**E**ference-based hierarchical reinforcement learning via hindsight **R**elabeling. The key realization underlying PIPER is that a suitably designed preference-based RL approach can be used to learn a high-level reward model that is simultaneously decoupled from non-stationary, lower-level rewards and carefully tailored to generate feasible subgoals. To achieve this, several innovations are necessary. First, to overcome the problem of acquiring the human preference feedback needed to learn the higher-level reward model, we propose a goal-conditioned, sparse reward-based approach to replace human trajectory preferences. Second, to address the sample-inefficiency arising from the use of sparse rewards, we apply hindsight experience relabeling [1] to learn a denser, more informative higher-level reward model. Finally, to encourage our higher-level policies to predict subgoals achievable by lower-level policies, we propose a novel value-function regularization scheme that calibrates subgoal selection to the current lower-level policy's abilities. Taken together, these techniques form the core of PIPER (see Figure 1). We perform extensive experiments in complex, sparse-reward environments that empirically show that PIPER demonstrates impressive performance and consistently outperforms the baselines, as well as ablation studies illustrating the importance of each of PIPER's component techniques.

We summarize the main contributions of PIPER:

- We demonstrate that PIPER is able to mitigate non-stationarity in off-policy HRL, by employing preference-based feedback.
- Since collecting human preference feedback is impractical, we propose an alternative primitive-in-the-loop (PiL) scheme to determine preferences between trajectories without human feedback.
- Using primitive informed regularization, PIPER generates efficient subgoals, thus improving performance.
- We employ hindsight relabeling to improve sample efficiency in preference-based learning and deal with sparsity in sparse-reward scenarios.
- PIPER uses soft target updates to mitigate instability due to non-stationary reward model learned using preference-based feedback.

- PIPER achieves greater than 50% success rates in challenging, sparse-reward robotic environments, where most other hierarchical and non-hierarchical baselines fail to achieve any significant progress.

2 Related Work

Hierarchical Reinforcement Learning. HRL approaches [3, 7, 2, 33, 10] promise the intuitive benefits of improved exploration and temporal abstraction [29]. In goal-conditioned feudal hierarchical learning-based approaches [9, 38], the higher-level policy predicts subgoals for the lower-level primitive, which in turn tries to achieve them by executing primitive actions on the environment. However, such off-policy HRL approaches face non-stationarity due to dynamically changing lower-level primitive behavior. Prior approaches [28, 24] partially address this non-stationarity by relabeling the goals in the replay buffer. In contrast, we propose a novel approach that first uses preference-based learning [6, 21] to learn a reward model, and subsequently uses the reward model to relabel the non-stationary rewards in the replay buffer.

The option learning framework [37, 18] provides the benefits of temporal abstraction by learning extended macro actions. Unfortunately, such approaches can lead to degenerate solutions and thus require additional regularization approaches to ensure feasible subgoal prediction. Notably, such degenerate solutions result in unbalanced task-split which nullifies the advantages of hierarchical learning. In our approach, we perform primitive-informed regularization to regularize the higher-level policy to predict achievable subgoals for lower primitive. Another line of work uses previously hand-designed action primitives [31, 7] to accelerate hierarchical learning. Such approaches, however, depend on the quality of hand-designed primitives, which are hard to design in complex tasks.

Preference-based Learning. Several prior works have proposed approaches that perform reinforcement learning on human rankings [19, 34, 40, 8]. [39] extend the TAMER framework by replacing the reward function by feedback signals. [6] used deep RL advancements to learn a reward model using neural networks, based on human preference feedback. Although some works employed on-policy RL [6] for solving tasks, a more sample efficient approach [21] used off-policy RL [13] to learn a policy using preference feedback, thereby improving sample efficiency. In this work, we propose an off-policy HRL method that uses preference-based feedback to mitigate non-stationarity, and uses primitive informed regularization to generate achievable subgoals for the lower primitive.

3 Problem Formulation

We consider an MDP (S, A, p, r, γ), where S is the state space, A is the action space, $p : S \times A \to \mathcal{P}(S)$ is the transition probability function mapping state-action pairs to probability distributions over the state space, $r : S \times A \to \mathbb{R}$ is the reward function, and $\gamma \in (0, 1)$ is the discount factor. At a given timestep t, the agent is in state s_t, takes action $a_t \sim \pi(a_t|s_t)$ according to some policy $\pi : S \to \mathcal{P}(A)$ which maps states to probability distributions over the action space, receives reward $r_t = r(s_t, a_t)$, and the system transitions to a new state $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$. The standard RL objective is given by

$$\pi^* := \arg\max_\pi J(\pi) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t r_t\right], \tag{1}$$

which is also called policy optimization in literature. The state action value function $Q(s, g, a)$ computes the expected cumulative reward when the start state is s, the final goal is g, and the next primitive action is a. In hierarchical RL (HRL) problem, the higher-level policy aims to achieve an end goal by issuing subgoals to the lower-level policy, while the lower-level policy chooses primitive actions oriented towards achieving the specified subgoals. In HRL, the higher-level policy $\pi^H : S \to \mathcal{P}(G)$ specifies a subgoal $g_t \in G$, where $G \subseteq S$ is the set of possible goals. During execution at each time step t, the subgoal $g_t \sim \pi^H(s_t)$ after every k timesteps and $g_t = g_{k\lfloor t/k \rfloor}$, otherwise. The effect of this is that the higher-level policy issues new subgoals every k timesteps and keeps subgoals fixed in between.

Furthermore, at each t, the lower-level policy $\pi^L : S \times G \to \mathcal{P}(A)$ selects primitive actions $a_t \sim \pi^L(s_t, g_t)$ according to the current state and subgoal specified by π^H, and the state transitions to $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$. Finally, at each timestep t, the higher level of the hierarchy provides the lower

level with reward $r_t^L = r^L(s_t, g_t, a_t) = -\mathbb{1}_{\{\|s_t - g_t\|_2 > \varepsilon\}}$, where $\mathbb{1}_B$ is the indicator function on a given set B, while the higher level receives reward $r_t^H = r^H(s_t, g, g_t)$, where $g \in \mathcal{G}$ is the end goal and $r^H : \mathcal{S} \times \mathcal{G} \times \mathcal{G} \to \mathbb{R}$ is a high-level reward function that we have yet to specify. The lower level populates its replay buffer with samples $(s_t, g_t, a_t, r_t^L, s_{t+1})$, while, at each t such that after every k timesteps, the higher level populates its buffer with samples of the form $(s_t, g, g_t, \sum_{i=t}^{t+k-1} r_i^H, s_{t+k})$. Next, we highlight key limitations of existing HRL methods. We provide the limitations of existing approaches to HRL in Appendix A.5

4 Proposed Approach

In this section, we introduce our approach, PIPER: Primitive-Informed Preference-based hierarchical reinforcement learning via hindsight Relabeling for solving complex sparse-reward tasks. The motivating idea behind our approach is that preference-based RL, where human preferences are used to learn a dense, informative reward function, can be used to learn a high-level reward function that simultaneously mitigates the issues of reward non-stationarity and infeasible subgoal generation described in Appendix A.5. Before presenting the main idea, let us brie y discuss preference-based learning.

4.1 Preference-based Learning (PBL)

In traditional RL, agents learn to maximize the accumulated reward $r(s_t, a_t)$ obtained from the environment, where the reward $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is assumed to be known. In many real-world scenarios, however, suitable environment rewards are notoriously dif cult to construct and typically require domain-speci c knowledge [32]. To learn the high-level reward necessary for the HRL setting considered in this paper, we use a preference-based RL setup, where the agent learns to perform the high-level task using preferences over agent behaviors [41, 6, 21, 15].

In the preference-based setting, agent behavior over a length k trajectory can be represented by a sequence, τ, of state observations and actions, $\tau = ((s_t, a_t), (s_{t+1}, a_{t+1}), \ldots, (s_{t+k-1}, a_{t+k-1}))$. The goal is to learn a reward function, $r_\psi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, with neural network parameters ψ, such that preferences between any two trajectory segments, τ^1, τ^2 can be modeled using the Bradley-Terry model [3]:

$$P_\psi[\tau^1 \succ \tau^2] = \frac{\exp \sum_t r_\psi(s_t^1, a_t^1)}{\sum_{i \in \{1,2\}} \exp \sum_t r_\psi(s_t^i, a_t^i)}, \tag{2}$$

where $\tau^1 \succ \tau^2$ denotes the event that τ^1 is preferred over τ^2. To learn parameters such that (2) matches the true preferences, behavior and preference data are recorded in a dataset $\mathcal{D}$ with entries of the form (τ^1, τ^2, y), where $y = (1, 0)$ when τ^1 is preferred over τ^2, $y = (0, 1)$ when τ^2 is preferred over τ^1, and $y = (0.5, 0.5)$ when there is no preference. The standard approach in the preference-based literature (see, e.g., [21]), which we adopt in PIPER, is to learn the reward function r_ψ using a cross-entropy loss:

$$L(\psi) = -\sum_{\mathcal{D}} y_1 \log P_\psi[\tau^1 \succ \tau^2] + y_2 \log P_\psi[\tau^2 \succ \tau^1], \tag{3}$$

where $(\tau^1, \tau^2, y) \in \mathcal{D}$ and y_1 and y_2 denote the first and second entries of y. Given the preference model and learning objective in (2) and (3), we now turn to the problem of acquiring preference data in our hierarchical setting. Additionally, We provide challenges of directly applying PBL in HRL in Appendix A.6

4.2 PiL: Primitive-in-the-Loop Feedback for HRL

We next introduce our Primitive-in-the-Loop (PiL) approach for generating higher-level preference feedback. This technique provides an effective, easily computable replacement for human feedback, avoiding the impracticality of obtaining human preference data for trajectories observed during training in the HRL setting. The key idea is to replace human preferences over higher-level trajectories with preferences generated by using a simple, goal-conditioned sparse reward to perform pairwise comparisons. We note that the sparsity of the proposed reward presents additional issues, which we address in Section 4.3 below.

Implicit reward functions: Let s be the current state, g be the subgoal predicted by higher level, and g be the final goal. In our goal-conditioned HRL setting, we represent k-length trajectories of lower-level primitive behavior as sequences given by $= ((s_t; a_t); (s_{t+1}; a_{t+1}); : : : ; (s_{t+k-1}; a_{t+k-1}))$, and n-length trajectories of higher-level behavior as sequences of states and subgoal predictions, i.e., $= ((s_t; g_t); (s_{t+k}; g_{t+k}); : : : ; (s_{t+(n-1)k}; g_{t+(n-1)k}))$, where the higher-level policy is executed for n timesteps and the lower-level policy is executed for k timesteps between each pair of consecutive subgoal predictions. In preference-based learning, the preferences are assumed to correspond to an implicit reward function $r : S \times G \times A \to \mathbb{R}$ implied by the true preferences. Let g be the final goal and ^{1} and ^{2} be two higher-level trajectories. Whenever ^{1} is preferred to ^{2}, denoted by $^1 \succ ^2$, then the assumption that encodes the true preferences implies:

$$\sum_{i=0}^{n-1} r(s^1_{t+ik}; g; g^1_{t+ik}) > \sum_{i=0}^{n-1} r(s^2_{t+ik}; g; g^2_{t+ik}) \tag{4}$$

Replacing human feedback: In the standard preference-learning framework, preferences are elicited from human feedback [6] and are subsequently used to learn the reward model. In this work, we replace this Human-in-the-Loop (HiL) feedback with Primitive-in-the-Loop (PiL) feedback by using implicit sparse rewards $r^s(s_t; g; g_t)$, defined presently, to determine preferences between behavior sequences ^{1} and ^{2}. We call this feedback Primitive-in-the-Loop since we generate this feedback using primitive sparse rewards. We obtain these primitive rewards as follows. Suppose the higher level policy ^{H} predicts subgoal $g_t \sim ^H(\cdot|s_t; g)$ for state s_t and goal g. The lower-level primitive executes primitive actions according to its policy ^{L} for k timesteps and ends up in state s_{t+k-1}. We use the sparse reward provided by the environment at state s_{t+k-1} as the implicit sparse reward, i.e., $r^s(s_t; g; g_t) = -\mathbb{1}_{\{s_{t+k-1} \neq g \ \wedge \ k<c\}}$. Note that this reward is directly available from the environment. We replace the implicit reward in Equation (4) with r^s, and thus use r^s to obtain preferences between higher level behavior sequences. Concretely, for the goal g, goal $^1 \succ ^2$ implies

$$\sum_{i=0}^{n-1} r^s(s^1_{t+ik}; g; g^1_{t+ik}) > \sum_{i=0}^{n-1} r^s(s^2_{t+ik}; g; g^2_{t+ik}) \tag{5}$$

The preferences elicited using r^s are subsequently used to learn the preference reward function.

Effect on non-stationarity: Off-policy HRL approaches suffer from non-stationarity due to outdated transitions in the higher-level replay buffer caused by the changing lower-level policy. In our approach, the rewards in the higher-level replay buffer are relabeled using r^s. Hence, the transitions are updated from $(s_t; g; g_t; \sum_{i=t}^{t+k-1} r_i; s_{t+k-1})$ to $(s_t; g; g_t; r^s(s_t; g; g_t); s_{t+k-1})$. Since r^s does not depend on changing lower primitive behavior, this reward relabeling eliminates the non-stationarity typically encountered in off-policy HRL.

4.3 PiL with Goal-conditioned Hindsight Relabeling

Despite the intuitive appeal of the sparse PiL feedback reward r^s proposed in Section 4.2 for providing preferences between higher-level trajectories, due to its sparsity it mostly fails to generate a meaningful reward signal. As a result, learning an appropriate reward function using r^s as our PiL primitive is unreliable. To address this issue, we employ hindsight relabeling [1] when comparing trajectories ^{1} and ^{2}. Specifically, we first randomly sample a new goal g' from the set $\{s^1_{t+ik}; s^2_{t+ik}\}_{i=1}^{n-1}$ of states encountered during trajectories ^{1}, ^{2}, then apply Equation (5) with g replaced by g':

$$\sum_{i=0}^{n-1} r^s(s^1_{t+ik}; g'; g^1_{t+ik}) > \sum_{i=0}^{n-1} r^s(s^2_{t+ik}; g'; g^2_{t+ik}): \tag{6}$$

Using hindsight relabeling, we are able to generate significantly better preference feedback, resulting in improved sample efficiency and performance during training. We show in Section 5, Figure 4 that this simple hindsight relabeling approach significantly boosts performance in complex sparse-reward tasks.

4.4 Primitive-informed Regularization

As discussed above, the preference reward learned using PiL with hindsight relabeling motivates the higher-level policy to reach the goal g while mitigating non-stationarity. However, the higher-level subgoal predictions g_t may be too difficult for the current lower-level policy, which may stall learning at the lower level (see Figure 2 for a comparison of PIPER with and without regularization). Ideally, the higher-level policy should produce subgoals at an appropriate level of difficulty, according to the current capabilities of the lower primitive. Properly balancing the task split between hierarchical levels is a recurring challenge in HRL.

For a given lower-level policy $\pi^L : S \times G \to A$, denote the corresponding state value function by $V^L(s; g)$. To encourage appropriate subgoal selection by the higher-level policy, we propose using the lower-level state value function V^L to regularize the higher-level policy to predict feasible subgoals for the lower-level policy. Intuitively, $V^L(s_t; g_t)$ provides an estimate of the achievability of subgoal g_t from current state s_t, since a high value of $V^L(s_t; g_t)$ implies that the lower level expects to achieve high reward for subgoal g_t. Let $r^s(s; g; g)$ denote the parameterized reward model corresponding to the preference data, as defined in Section 4.2. For a trajectory of length T, consider the following KL-regularized formulation of preference-based learning, where π^{reg} is the regularizing policy for the higher-level policy π^H:

$$\max_{\pi^H} \mathbb{E}_{\pi^H}\Big[\sum_{t=0}^{T} \big(r^s(s; g; g) - \alpha D_{KL}[\pi^H(\cdot|s_t) \| \pi^{reg}(\cdot|s_t)]\big)\Big] \tag{7}$$

where $\alpha \geq 0$ is a scalar hyperparameter controlling the deviation from the regularization policy π^{reg}. We propose the following formulation of the regularization policy:

$$\pi^{reg}(g_t \mid s_t) = \frac{\exp(m(V^L(s_t; g_t)))}{\hat{Z}(s_t)} \tag{8}$$

where $\hat{Z}(s_t) = \sum_{g_t} \exp(m(V^L(s_t; g_t))$, and $m = \frac{1}{\alpha}$. The policy $\pi^{reg}(\cdot|g_t)$ is simply the softmax distribution generating a given subgoal g_t with probability proportional to its value $V^L(s_t; g_t)$. We substitute (8) in (7) to get

$$\max_{\pi^H} \mathbb{E}_{\pi^H}\Big[\sum_{t=0}^{T} \big(r^s(s; g; g) + \alpha(V^L(s_t; g_t)) + \alpha\hat{m}(s_t)\big)\Big]; \tag{9}$$

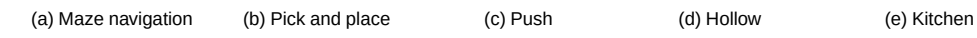
where $\hat{m}(s_t) = -H(s_t) - \log \hat{Z}(s_t)$, and $H(s_t) = -\log \pi^H(g_t|s_t)$ is the entropy term for π^H. Following prior work [22, 42], we get the following optimal solution for the higher-level policy:

$$\pi^H(g_t|s_t) = \frac{1}{Z(s_t)} \exp\Big(\frac{1}{\alpha}(r^s(s; g; g) + \alpha(V^L(s_t; g_t)))\Big); \tag{10}$$

where $Z(s_t) = \sum_{g_t} \exp(\frac{1}{\alpha}(r^s(s; g; g) + \alpha(V^L(s_t; g_t))))$ is the partition function and α is the primitive regularization weight hyperparameter. Appendix A.1 contains the complete derivation. Notice that this optimal policy $\pi^H(g_t|s_t)$ assigns high probability to subgoals g_t which maximize the regularized reward $r^{total}(s; g; g) = r^s(s; g; g) + \alpha(V^L(s_t; g_t))$. If we use r^{total} to generate preferences between trajectories instead of the standard preferences outlined in Section 4.1, we end up with the optimal policy under this primitive-regularized, preference-based learning scheme. Hence, we substitute $r^{total}(s; g; g)$ into inequality (6) to yield our final preference condition for determining if $\pi^1 \succ \pi^2$:

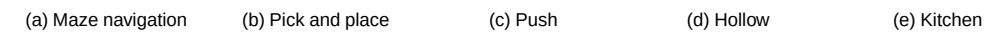
$$\sum_{i=0}^{n-1} r^{total}(s^1_{t+ik}; g; g^1_{t+ik}) > \sum_{i=0}^{n-1} r^{total}(s^2_{t+ik}; g; g^2_{t+ik}). \tag{11}$$

PIPER uses the preferences elicited by (11) to learn the reward function r^s, which is in turn used to perform reward relabeling of the higher-level replay buffer. As illustrated in Figure 2, value regularization can lead to significantly improved performance in certain tasks, while leading to minimal performance degradation in others.



(a) Maze navigation (b) Pick and place (c) Push (d) Hollow (e) Kitchen

Figure 2: Success rate comparison. This figure compares the success rate performances on four sparse maze navigation and robotic manipulation environments. The solid line and shaded regions represent the mean and standard deviation across 5 seeds. We compare our approach PIPER against multiple baselines. As can be seen, PIPER shows impressive performance and significantly outperforms the baselines.



(a) Maze navigation (b) Pick and place (c) Push (d) Hollow (e) Kitchen

Figure 3: Learning rate ablation. This figure compares the success rate performances for various values of primitive informed regularization weight hyper-parameter. If is too small, we loose the advantages of primitive informed regularization, leading to degrading performance. In contrast, if is too large, it may lead to degenerate solutions. Thus, these success rate performance plots demonstrate that proper primitive subgoal regularization is crucial for appropriate subgoal prediction, and improving overall performance.

4.5 PIPER Implementation

Reward stabilization using target networks: To mitigate potential training instability when the reward model is learned using preference-based learning, we utilize target networks with soft target updates [25]. In practice, we found this to greatly stabilize learning.

Pseudocode details: We explain our approach in detail in Appendix A.2 Algorithm 1. The higher- and lower-level policies are both trained using Soft Actor Critic (SAC) [13]. The rewards in higher level transitions are relabeled using the reward model. Instead of relabeling all replay buffer transitions, we relabel the higher-level buffer transitions as they are sampled.

5 Experiments

We perform experiments to empirically investigate the following questions: (1) How well does PIPER perform in sparse maze navigation and robotic manipulation tasks? (2) Is PIPER able to mitigate the recurring issue of non-stationarity in HRL? (3) Does PIPER outperform at preference-based learning? (4) Does PIPER enhance sample efficiency and training stability? (5) What is the contribution of each of the design choices in PIPER?

Setup: We evaluate PIPER on four robotic navigation and manipulation tasks: (i) maze navigation, (ii) pick and place [1], (iii) push, (iv) hollow, and (v) franka kitchen [12]. Since we focus on sparse reward scenarios, we re-implement these environments as sparse reward environments. Notably, since the pick and place, push, hollow and kitchen task environments are difficult, in order to speedup training, we assume access to a single human demonstration, and use an additional imitation learning objective at the lower level. We do not assume access to any demonstration in the maze navigation task. We keep this assumption consistent among all baselines to ascertain fair comparisons.

For environments (i) - (v) ((i) maze navigation, (ii) pick and place [1], (iii) push, (iv) hollow, and (v) franka kitchen [12]), the maximum task horizon is set to 225, 50, 50, 100, 225 timesteps, respectively, and the lower primitive is allowed to execute for = 15; 7; 7; 10 and 15 timesteps, respectively. In our experiments, we use off-policy Soft Actor Critic (SAC) [13] for optimizing RL objective, using the Adam [17] optimizer. The actor and critic networks are formulated as three-layer, fully connected neural networks with 512 neurons in each layer. The experiments are run for 6.75e5, 1.5e5, 6.5E5, and 6.75e5 timesteps in environments (i) - (iv), respectively. In our experiments,

Figure 4: Hindsight Relabeling ablation. This figure compares the performance of our PIPER approach with PIPER-No-HR ablation, which is effectively PIPER without hindsight relabeling [1] (as explained in Section 4.3). The plots showcase that although hindsight relabeling demonstrates minor performance improvement in sparse maze and kitchen tasks, it provides significant training speedup in sparse pick and place and push environments.

Figure 5: Target networks ablation. This figure compares the performance of our PIPER approach with PIPER-No-Target ablation, which is effectively PIPER without target networks implementation [25]. The plots showcase that using target networks significantly improves performance and indeed reduces training instability caused by non-stationary reward models, as discussed in [25] using preference based learning.

we use off-policy Soft Actor Critic (SAC) [13] for optimizing RL objective, using the Adam [17] optimizer. We provide additional implementation details in Appendix A.4.

Evaluation and Results. In order to analyse our design choices, we consider multiple baselines: PIPER-No-V (PIPER without primitive informed regularization, Section 4.4), RFLAT (Single-level PEBBLE [6] implementation with target networks), HIER (vanilla hierarchical SAC implementation), HAC (Hindsight Actor Critic [24]), DAC (Discriminator Actor Critic [20]), FLAT (Single-level SAC), and RAPS [7]. In Figure 2, we compare the success rate performances of PIPER with hierarchical and non-hierarchical baselines. To illustrate the advantage of primitive informed regularization, we implemented PIPER-No-V, which is PIPER without primitive-informed regularization. As seen in Figure 2, though PIPER significantly outperforms PIPER-No-V baseline in pick and place and push tasks, it only slightly outperforms PIPER-No-V in the maze, hollow and kitchen environments. This illustrates that primitive-informed regularization successfully encourages the higher-level policy to generate subgoals achievable by lower-level policies.

We implemented two other baselines: HAC (Hierarchical Actor Critic) [1], a hierarchical approach that deals with the non-stationarity issue by relabeling transitions, while assuming an optimal lower primitive; and HIER, which is a vanilla HRL baseline implemented using SAC. As seen in Figure 2, PIPER is able to significantly outperform both these baselines, illustrating that PIPER is indeed able to mitigate non-stationarity in HRL. We also compare PIPER with RFLAT, which is a re-implementation of PEBBLE [6] with target networks, and where the preference feedback is generated using PiL approach. Since the environments are sparse, we also augmented PEBBLE with hindsight relabeling in our implementation of RFLAT. As seen in Figure 2, PIPER is able to significantly outperform RFLAT. This empirically illustrates that our hierarchical preference-based learning approach is able to outperform at preference-based learning approach.

We finally compare against RAPS [7], a hierarchical approach where the higher-level policy picks from among hand-designed action primitives at the lower level. Importantly, the performance of RAPS depends on the quality of action primitives. We found that RAPS was able to outperform PIPER and all other baselines in the maze navigation environment. We hypothesize that this is because the hand-designed lower primitive used in the maze task makes the task significantly easier for the higher level. However, RAPS did not show progress in the more difficult pick and place, push, hollow and kitchen tasks. We also implemented Discriminator Actor-Critic (DAC) [20] with access to a single demonstration in the pick and place, push, hollow and kitchen environments, to analyse how PIPER fares against single-level baselines trained with privileged information. Additionally, we

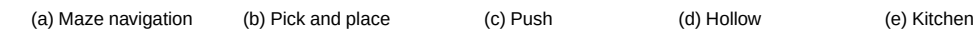


Figure 6: Dense rewards ablation. This figure compares the success rate performances between PIPER and two baselines based on dense rewards: (i) SAC-Dense, and (ii) PEBBLE-Dense. Although SAC-Dense outperforms PIPER on easier tasks, PIPER is able to outperform SAC-Dense in harder tasks. Notably, PEBBLE-Dense is unable to solve any of the tasks. Thus, PIPER shows impressive performance, and is a viable approach in complex sparse reward scenarios.

compared PIPER with a single-level RL baseline implemented using SAC. Both baselines failed to show significant progress, while PIPER clearly outperforms the baselines.

Ablation Analysis. Here, we empirically demonstrate the importance of our each various component technique, by performing corresponding ablation study. We analyze the effect of varying the weight parameter α in Figure 3, and found that setting appropriate value of hyperparameter α is crucial for improved performance. If α is too small, we lose the advantages of primitive-informed regularization, leading to poor performance. In contrast, if α is too large, it leads to degenerate solutions. We also analyze the effect of removing the hindsight relabeling (PIPER-No-HR ablation). In Figure 4, the empirical results show that hindsight relabeling indeed leads to improved performance. Further, we remove the target networks from PIPER (PIPER-No-Target), to analyze whether using target networks stabilizes training. As seen in Figure 5, target networks significantly improve training stability. Finally, in Figure 6, we compare PIPER against two baselines with well-behaved, hand designed dense reward functions (i) at SAC with dense reward function (SAC-Dense), and (ii) PEBBLE with actual human preferences (PEBBLE-Dense)). This comparison aims to aid the practitioners for choosing between hand-designing dense reward functions, or implementing PIPER. Although SAC-Dense outperforms PIPER in easier tasks, PIPER is able to outperform SAC-Dense in harder environments. Notably, SAC-Dense completely fails to solve the kitchen task. PEBBLE-Dense is unable to show good performance in any task. We provide further discussion in Appendix A.7.1.

6 Discussion

Limitations: PIPER uses L2 distance between states as the informative metric, which might be hard to compute in scenarios where the subgoals and goals are high-dimensional (e.g. images). A possible way to deal with this is to find a near optimal latent representation [27]. Further, although PIPER replaces human-in-the-loop preferences via primitive-in-the-loop (PiL) approach to generate trajectory preferences, human preferences may contain additional information that PiL may be unable to capture (say, humans may prefer trajectories that are safe to traverse, e.g. which do not involve any wall collisions). A possible way to deal with this issue is to employ additional rewards in Eqn 5, which prefer a safe trajectory. Similarly, the proposed primitive regularization approach may not generate such safe subgoals in its naive form. To this end, we can additionally learn a safety value function $V^s(s, g)$, which regularizes the higher agents to produce safe subgoals. Finally, the proposed target networks require additional memory and training overhead, and setting the hyper-parameter τ. However, we found in practice that the memory and training overheads are minimal, and setting τ does not cause a significant overhead. We would like to overcome these limitations in future work.

Conclusion and future work: In this work, we propose PIPER, a primitive informed hierarchical RL algorithm that leverages preference based learning to mitigate the non-stationarity issue in HRL. Using primitive informed regularization, PIPER is able to generate efficient subgoals, according to the goal achieving capability of the lower primitive. We demonstrate that by incorporating primitive informed regularization, hindsight relabeling, and target networks, PIPER is able to solve complex robotic tasks, and significantly outperform the baselines. Additionally, PIPER is able to outperform at preference based learning. We therefore believe that hierarchical preference based learning is a promising step towards building practical robotic systems that solve complex real-world tasks.

References

[1] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. CoRR, abs/1707.01495, 2017.

[2] Andrew G. Barto and Sridhar Mahadevan. Recent advances in hierarchical reinforcement learning. Discrete Event Dynamic Systems, 13:341–379, 2003.

[3] Ralph Allan Bradley and Milton E. Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. Biometrika, 39:324, 1952.

[4] Zehong Cao, Kaichiu Wong, and Chin-Teng Lin. Human preference scaling with demonstrations for deep reinforcement learning. arXiv preprint arXiv:2007.12904, 2020.

[5] Elliot Chane-Sane, Cordelia Schmid, and Ivan Laptev. Goal-conditioned reinforcement learning with imagined subgoals. In International Conference on Machine Learning, pages 1430–1440. PMLR, 2021.

[6] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. Advances in neural information processing systems, 30, 2017.

[7] Murtaza Dalal, Deepak Pathak, and Russ R Salakhutdinov. Accelerating robotic reinforcement learning via parameterized action primitives. Advances in Neural Information Processing Systems, 34:21847–21859, 2021.

[8] Christian Daniel, Oliver Kroemer, Malte Viering, Jan Metz, and Jan Peters. Active reward learning with a novel acquisition function. Autonomous Robots, 39:389–405, 2015.

[9] Peter Dayan and Geoffrey E Hinton. Feudal reinforcement learning. Advances in neural information processing systems, 5, 1992.

[10] Thomas G. Dietterich. Hierarchical reinforcement learning with the MAXQ value function decomposition. CoRR, cs.LG/9905014, 1999.

[11] Shixiang Gu, Ethan Holly, Timothy P. Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation. CoRR, abs/1610.00633, 2016.

[12] Abhishek Gupta, Vikash Kumar, Corey Lynch, Sergey Levine, and Karol Hausman. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. arXiv preprint arXiv:1910.11956, 2019.

[13] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. CoRR, abs/1801.01290, 2018.

[14] Jean Harb, Pierre-Luc Bacon, Martin Klissarov, and Doina Precup. When waiting is not an option: Learning options with a deliberation cost. In Proceedings of the AAAI Conference on Arti cial Intelligence, volume 32, 2018.

[15] Borja Ibarz, Jan Leike, Tobias Pohlen, Geoffrey Irving, Shane Legg, and Dario Amodei. Reward learning from human preferences and demonstrations in atari, 2018.

[16] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. CoRR, abs/1806.10293, 2018.

[17] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

[18] Martin Klissarov, Pierre-Luc Bacon, Jean Harb, and Doina Precup. Learnings options end-to-end for continuous action tasks. arXiv preprint arXiv:1712.00004, 2017.

[19] W Bradley Knox and Peter Stone. Interactively shaping agents via human reinforcement: The tamer framework. In Proceedings of the fth international conference on Knowledge capture pages 9–16, 2009.

[20] Ilya Kostrikov, Kumar Krishna Agrawal, Debidatta Dwibedi, Sergey Levine, and Jonathan Tompson. Discriminator-actor-critic: Addressing sample inef ciency and reward bias in adversarial imitation learning arXiv preprint arXiv:1809.02925, 2018.

[21] Kimin Lee, Laura Smith, and Pieter Abbeel. Pebble: Feedback-ef cient interactive reinforcement learning via relabeling experience and unsupervised pre-training, 2021.

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

[23] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies CoRR, abs/1504.00702, 2015.

[24] Andrew Levy, George Konidaris, Robert Platt, and Kate Saenko. Learning multi-level hierarchies with hindsight. In International Conference on Learning Representations, 2018.

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

[26] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. CoRR, abs/1312.5602, 2013.

[27] O r Nachum, Shixiang Gu, Honglak Lee, and Sergey Levine. Near-optimal representation learning for hierarchical reinforcement learning arXiv preprint arXiv:1810.01257, 2018.

[28] O r Nachum, Shixiang Shane Gu, Honglak Lee, and Sergey Levine. Data-ef cient hierarchical reinforcement learning Advances in neural information processing systems, 31, 2018.

[29] O r Nachum, Haoran Tang, Xingyu Lu, Shixiang Gu, Honglak Lee, and Sergey Levine. Why does hierarchy (sometimes) work so well in reinforcement learning? arXiv preprint arXiv:1909.10618, 2019.

[30] Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming exploration in reinforcement learning with demonstrations 2018 IEEE international conference on robotics and automation (ICRA), pages 6292–6299. IEEE, 2018.

[31] Soroush Nasiriany, Huihan Liu, and Yuke Zhu. Augmenting reinforcement learning with behavior primitives for diverse manipulation tasks CoRR, abs/2110.03655, 2021.

[32] Andrew Y Ng, Daishi Harada, and Stuart Russell. Policy invariance under reward transformations: Theory and application to reward shaping Icml, volume 99, pages 278–287. Citeseer, 1999.

[33] Ronald Parr and Stuart Russell. Reinforcement learning with hierarchies of machines. In M. Jordan, M. Kearns, and S. Solla, editors, Advances in Neural Information Processing Systems, volume 10. MIT Press, 1998.

[34] Patrick M Pilarski, Michael R Dawson, Thomas Degris, Farbod Fahimi, Jason P Carey, and Richard S Sutton. Online human training of a myoelectric prosthesis controller via actor-critic reinforcement learning. In 2011 IEEE international conference on rehabilitation robotics, pages 1–7. IEEE, 2011.

[35] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations CoRR, abs/1709.10087, 2017.

[36] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. nature, 529(7587):484–489, 2016.

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

[38] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In International Conference on Machine Learning, pages 3540–3549. PMLR, 2017.

[39] Garrett Warnell, Nicholas Waytowich, Vernon Lawhern, and Peter Stone. Deep tamer: Interactive agent shaping in high-dimensional state spaces. In Proceedings of the AAAI conference on arti cial intelligence, volume 32, 2018.

[40] Aaron Wilson, Alan Fern, and Prasad Tadepalli. A bayesian approach for policy learning from trajectory preference queries. Advances in neural information processing systems, 25, 2012.

[41] Aaron Wilson, Alan Fern, and Prasad Tadepalli. A bayesian approach for policy learning from trajectory preference queries. In F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger, editors, Advances in Neural Information Processing Systems, volume 25. Curran Associates, Inc., 2012.

[42] Brian D Ziebart, Andrew L Maas, J Andrew Bagnell, Anind K Dey, et al. Maximum entropy inverse reinforcement learning. Aaai, volume 8, pages 1433–1438. Chicago, IL, USA, 2008.

A Appendix

A.1 Deriving the final optimum of KL-Constrained Reward Maximization Objective

In this appendix, we will derive Eqn 10 from Eqn 7. Thus, we optimize the following objective:

$$\max_{\pi} E_{\pi}[\sum_{t=0}^{T} (r^{s}(s; g_t; g) - D_{KL}[\pi^{H}(g_t|s_t) \| \pi_{reg}(g_t|s_t)])] \tag{12}$$

Re-writing the above equation after expanding KL divergence formula:

$$= \max_{\pi} E_{\pi}[\sum_{t=0}^{T} (r^{s}(s; g_t; g) - \log \frac{\pi^{H}(g_t|s_t)}{\pi_{reg}(g_t|s_t)})] \tag{13}$$

$$= \max_{\pi} E_{\pi}[\sum_{t=0}^{T} (r^{s}(s; g_t; g) - \log \pi^{H}(g_t|s_t) + \log \pi_{reg}(g_t|s_t))] \tag{14}$$

Substituting π_{reg} from Eqn 8, and $m = -$ in Equation 14,

$$= \max_{\pi} E_{\pi}[\sum_{t=0}^{T} (r^{s}(s; g_t; g) - \log \pi^{H}(g_t|s_t) + \log \exp(m(V^{\pi_L}(s_t; g_t))) - \log \sum_{g_t} \exp(m(V^{\pi_L}(s_t; g_t))))] \tag{15}$$

$$= \max_{\pi} E_{\pi}[\sum_{t=0}^{T} (r^{s}(s; g_t; g) - \log \pi^{H}(g_t|s_t) + (V^{\pi_L}(s_t; g_t)) - \log \sum_{g_t} \exp(m(V^{\pi_L}(s_t; g_t))))] \tag{16}$$

$$= \min_{\pi} E_{\pi}[\sum_{t=0}^{T} (\log \pi^{H}(g_t|s_t) - \frac{1}{\beta}(r^{s}(s; g_t; g) + (V^{\pi_L}(s_t; g_t))) + \log \sum_{g_t} \exp(m(V^{\pi_L}(s_t; g_t))))] \tag{17}$$

$$= \min_{\pi} E_{\pi}[\sum_{t=0}^{T} (\log(\frac{\pi^{H}(g_t|s_t)}{\exp(\frac{1}{\beta}(r^{s}(s; g_t; g) + (V^{\pi_L}(s_t; g_t))))}) + \log \sum_{g_t} \exp(m(V^{\pi_L}(s_t; g_t))))] \tag{18}$$

$$= \min_{\pi} E_{\pi}[\sum_{t=0}^{T} (\log(\frac{\pi^{H}(g_t|s_t)}{\frac{1}{Z(s_t)} \exp(\frac{1}{\beta}(r^{s}(s; g_t; g) + (V^{\pi_L}(s_t; g_t))))}) + \log \sum_{g_t} \exp(m(V^{\pi_L}(s_t; g_t))) - \log Z(s_t))] \tag{19}$$

where, $Z(s_t) = \sum_{g_t} \exp(\frac{1}{\beta}(r^{s}(s; g_t; g) + (V^{\pi_L}(s_t; g_t))))$

Note that the partition function $Z(s_t)$ and the term $\log \sum_{g_t} \exp(m(V^L(s_t, g_t)))$, do not depend on the policy π^U

$$= \min_{\pi^U} \mathbb{E}_{\pi^U}\Big[\sum_{t=0}^{T} \big(D_{KL}[\pi^H(g_t|s_t) \| \pi^H_\theta(g_t|s_t)] - \log \sum_{g_t} \exp(m(V^L(s_t, g_t))) - \log Z(s_t)\big)\Big] \quad (20)$$

where, $\pi^H_\theta(g_t|s_t) = \frac{1}{Z(s_t)} \exp(\frac{1}{\alpha}(r^S(s, g, g) + \gamma V^L(s_t, g_t)))$ which is a valid probability distribution. $\pi^H_\theta(g_t|s_t)$ is minimized when $D_{KL} = 0$. Hence,

$$\pi^H(g_t|s_t) = \pi^H_\theta(g_t|s_t) = \frac{1}{Z(s_t)} \exp(\frac{1}{\alpha}(r^S(s, g, g) + \gamma V^L(s_t, g_t))) \quad (21)$$

A.2 PIPER Algorithm

Algorithm 1 PIPER

Initialize preference dataset $\mathcal{D} = \{\}$
Initialize higher level replay buffer $R^H = \{\}$ and lower level replay buffer $R^L = \{\}$
Initialize reward model parameters
Initialize target reward model parameters θ'
for $i = 1 : : : N$ **do**
  // Collect experience using π^H and π^L and store transitions in R^H and R^L
  for each timestep **do**
    $d^H \leftarrow d^H \cup \{(s_t, g, g_t, \sum_{i=t}^{t+k-1} r_i, s_{t+k-1})\}$
    $d^L \leftarrow d^L \cup \{(s_t, g_t, a_t, r_t, s_{t+1})\}$
  $R^H \leftarrow R^H \cup d^H$
  $R^L \leftarrow R^L \cup d^L$
  // Sample higher level behavior trajectories and goal
  $(\tau^1, \tau^2) \sim R^H$ and $g \sim G$
  Sample a set of additional goals For relabeling $G' \leftarrow \{g'\}$
  for $g' \in G'$ **do**
    Relabel g by g' and generate label y using Equation (11)
    Store preference $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\tau^1, \tau^2, y)\}$
  // Reward Model Learning
  for each gradient step **do**
    Optimize model reward θ using Equation (3)
    // Soft update target reward model parameters
    $\theta' \leftarrow \rho\theta' + (1-\rho)\theta$
  // Policy Learning
  for each gradient step **do**
    Sample $\{(\tau_j)\}_{j=1}^m$ from R^H
    Sample $\{(\tau_j)\}_{j=1}^m$ from R^L
    Relabel rewards in $\{(\tau_j)\}_{j=1}^m$ using target reward model θ'
    Optimize higher policy π^H using SAC
    Optimize lower policy π^L using SAC

A.3 Implementation details

To ensure fair comparisons, we keep parameters including neural network layer width, number of layers, choice of optimizer, SAC implementation parameters, etc., consistent across all baselines. For environments (i) – (v), the primitive regularization weight hyper-parameter α is set as 10^{-5}, 10^{-4}, 10^{-4}, 10^{-4}, 10^{-6} in environments (i) – (v), respectively (see Figure 3). For implementing

the RAPS baseline, we use the following lower-level behaviors: in maze navigation, we design a single primitive,reach, where the lower-level primitive travels in a straight line towards the subgoal predicted by higher level. In the pick and place, push and hollow tasks, we design three primitives: gripper-reach, where the gripper goes to given position$(x_i; y_i; z_i)$; gripper-open, which opens the gripper; andgripper-close, which closes the gripper. In the kitchen environment, we use the action primitives implemented in RAPS [7].

In the maze navigation task, a7-degree-of-freedom (7-DoF) robotic arm navigates across a four room maze, where the closed gripper (xed at table height) has to navigate across the maze to the goal position. In pick and place task, a7-DoF robotic arm gripper has to navigate to the square block, pick it up and bring it to the goal position. In the push task, a7-DoF robotic arm gripper has to push the square block towards the goal position. In hollow task, a7-DoF robotic arm gripper has to pick a square hollow block and place it such that a xed vertical structure on the table goes through the hollow block. In the kitchen task, a9-DoF franka robot has to pre-de ne a complex task to achieve the nal goal. We consider the task where the robot has to open the microwave door. In addition, we compare our approach with Discriminator Actor-Critic [20], which is provided a single expert demonstration. Although not explored in this work, combining preference-based learning and learning from demonstrations is an interesting research avenue [4].

A.3.1 Additional hyper-parameters

Here, we enlist the additional hyper-parameters used in PIPER:

activation: tanh [activation for reward model]
layers: 3 [number of layers in the critic/actor networks]
hidden: 512 [number of neurons in each hidden layers]
Q_lr: 0.001 [critic learning rate]
pi_lr: 0.001 [actor learning rate]
buffer_size: int(1E7) [for experience replay]
tau: 0.8 [polyak averaging coefficient]
clip_obs: 200 [clip observation]
n_cycles: 1 [per epoch]
n_batches: 10 [training batches per cycle]
batch_size: 1024 [batch size hyper-parameter]
reward_batch_size: 50 [reward batch size for PEBBLE and RFLAT]
random_eps: 0.2 [percentage of time a random action is taken]
alpha: 0.05 [weightage parameter for SAC]
noise_eps: 0.05 [std of gaussian noise added to not-completely-random actions]
norm_eps: 0.01 [epsilon used for observation normalization]
norm_clip: 5 [normalized observations are cropped to this values]
adam_beta1: 0.9 [beta 1 for Adam optimizer]
adam_beta2: 0.999 [beta 2 for Adam optimizer]

A.4 Environment details

A.4.1 Maze navigation task

In this environment, a7-DOF robotic arm gripper navigates across random four room mazes. The gripper arm is kept closed and the positions of walls and gates are randomly generated. The table is discretized into a rectangular$W \times H$ grid, and the vertical and horizontal wall positionsW_P andH_P are randomly picked from$(1; W - 2)$ and$(1; H - 2)$ respectively. In the four room environment thus constructed, the four gate positions are randomly picked from$(1; W_P - 1)$, $(W_P + 1; W - 2)$, $(1; H_P - 1)$ and$(H_P + 1; H - 2)$. The height of gripper is kept xed at table height, and it has to navigate across the maze to the goal position(shown as red sphere).

The following implementation details refer to both the higher and lower level polices, unless otherwise explicitly stated. The state and action spaces in the environment are continuous. The state is represented as the vector$[p; M]$, where$p$ is current gripper position and$M$ is the sparse maze array. The higher level policy input is thus a concatenated vector$[p; M; g]$, where$g$ is the target

goal position, whereas the lower level policy input is concatenated vector $[s; M; s_g]$, where s_g is the sub-goal provided by the higher level policy. The current position of the gripper is the current achieved goal. The sparse maze array M is a discrete 2D one-hot vector array, where 1 represents presence of a wall block, and 0 absence. In our experiments, the size of M are kept to be 3 and 11 respectively. The upper level predicts subgoal s_g, hence the higher level policy action space dimension is the same as the dimension of goal space of lower primitive. The lower primitive action a which is directly executed on the environment, is a 4 dimensional vector with every dimension $a_i \in [0, 1]$. The first 3 dimensions provide offsets to be scaled and added to gripper position for moving it to the intended position. The last dimension provides gripper control (0 implies a fully closed gripper, 0.5 implies a half closed gripper and 1 implies a fully open gripper).

A.4.2 Pick and place, Push and Hollow Environments

In the pick and place environment, a 7-DOF robotic arm gripper has to pick a square block and bring/place it to a goal position. We set the goal position slightly higher than table height. In this complex task, the gripper has to navigate to the block, close the gripper to hold the block, and then bring the block to the desired goal position. In the push environment, 7-DOF robotic arm gripper has to push a square block towards the goal position. In hollow task, 7-DOF robotic arm gripper has to pick a square hollow block and place it such that a fixed vertical structure on the table goes through the hollow block. The state is represented as the vector $[p; o; q; e]$, where p is current gripper position, o is the position of the block object placed on the table, q is the relative position of the block with respect to the gripper, and e consists of linear and angular velocities of the gripper and the block object. The higher level policy input is thus a concatenated vector $[p; o; q; e; g]$, where g is the target goal position. The lower level policy input is concatenated vector $[p; o; q; e; s_g]$, where s_g is the sub-goal provided by the higher level policy. The current position of the block object is the current achieved goal. In our experiments, the sizes of p, o, q, e are kept to be 3, 3, 3 and 11 respectively. The upper level predicts subgoal s_g, hence the higher level policy action space and goal space have the same dimension. The lower primitive action a is a 4 dimensional vector with every dimension $a_i \in [0, 1]$. The first 3 dimensions provide gripper position offsets, and the last dimension provides gripper control (0 means closed gripper and 1 means open gripper). While training, the position of block object and goal are randomly generated (block is always initialized on the table, and goal is always above the table at a fixed height).

A.5 Limitations of Existing Approaches to HRL

While HRL promises significant advantages over non-hierarchical RL, including improved sample efficiency due to temporal abstraction and improved exploration [28, 29], serious challenges remain. In this work, we focus on two outstanding issues:

C1: the destabilizing effect of lower-level reward non-stationarity on off-policy HRL,

C2: and performance degradation due to infeasible subgoal generation by higher-level policies.

As discussed in [28] and [24], off-policy HRL suffers from non-stationarity due to changing lower primitive behavior, since, for a given transition in the higher-level replay buffer $(s_t; g; g_t; \sum_{i=t}^{t+k-1} r_i^H; s_{t+k})$, the rewards and transition may have been generated by an outdated lower-level policy. In addition, it was observed in [5] that, without taking care to ensure valid subgoal selection at the higher level, HRL methods may issue unachievable subgoals to the lower-level policy, stalling learning at the lower level and significantly impacting performance. The primary motivation of this work is the development of a novel, preference-based learning-inspired technique for learning the high-level reward function, r^H, that addresses these key limitations of existing HRL methods.

A.6 Challenges of directly applying PBL in HRL

- First, collecting the quantity of human feedback needed to enable preference-based learning of the high-level reward function is impractical. To address this, we propose a replacement PiL scheme to determine preferences between trajectories without human feedback.
- Second, standard goal-conditioned rewards are too sparse for sample-efficient learning. We therefore incorporate hindsight relabeling to reduce sparsity and increase the informativeness of observed trajectories.

- Third, even with hindsight relabeling the higher-level policy may choose subgoals that are infeasible for the lower-level policy, so we regularize the PiL reward with the value function of the lower-level policy to encourage feasible subgoal selection.
- Finally, we incorporate soft target updates [25] to mitigate the training instability while learning the high-level reward function. Taken together, this combination of techniques constitutes PIPER, the primary contribution of this paper. The rest of this section provides details on each aspect of our approach. Pseudocode for PIPER is provided in Appendix A.2.

A.7 Additional experiments

In this section, we provide additional experiments, comparing PIPER with additional baselines.

A.7.1 Experiments with dense rewards baselines

We compare the performance of these baselines with PIPER in Figure 6. We find that single level SAC implementation with explicitly hand-designed and fine-tuned dense rewards (which we call SAC-Dense baseline) outperforms PIPER in easier tasks like Maze navigation and Push environments. However, we find that as the task complexity increases, the performance of SAC-Dense degrades and PIPER is able to outperform SAC-Dense in Pick and Place and kitchen environments. In the kitchen environment, SAC-Dense fails to solve the task. We believe this to is due to two reasons:

- hand-designing a suitable reward in complex environments is comparatively hard, and may lead to sub-optimal performance, and
- the advantages of various design choices in PIPER (exploration and task abstraction due to hierarchical structure, mitigating non-stationarity using preference based RL, reward densification using hindsight relabeling, feasible subgoal generation using primitive regularization, and added training stability due to target networks) out-weigh the presence of dense rewards in SAC-Dense baseline.

We also implemented PEBBLE with actual human preferences (rather than primitives), but find that this baseline is unable to show good performance in any of the tasks. This shows that although PEBBLE shows impressive performance in simple environments, it needs further improvements to make it work in complex robotic manipulation tasks like the ones explored in this work.

A.8 Environment visualizations

Here, we provide some visualizations of the agent successfully performing the task.

Figure 7: Maze navigation task visualization The visualization is a successful attempt at performing maze navigation task

Figure 8: Pick and place task visualization This figure provides visualization of a successful attempt at performing pick and place task