A Training details

Goal distributions. We train our goal-conditioned value function, high-level policy, and low-level policy respectively with Equations (4), (6) and (7), using different goal-sampling distributions. For the value function (Equation (4)), we sample the goals from either random states, futures states, or the current state with probabilities of 0.3, 0.5, and 0.2, respectively, following Ghosh et al. [34]. We use $\mathrm{Geom}(1-\gamma)$ for the future state distribution and the uniform distribution over the offline dataset for sampling random states. For the hierarchical policies, we mostly follow the sampling strategy of Gupta et al. [35]. We first sample a trajectory $(s_0, s_1, \dots, s_t, \dots, s_T)$ from the dataset $\mathcal{D}_{\mathcal{S}}$ and a state s_t from the trajectory. For the high-level policy (Equation (6)), we either (*i*) sample g uniformly from the future states s_{t_g} $(t_g > t)$ in the trajectory and set the target subgoal to $s_{\min(t+k,t_g)}$ or (*ii*) sample g uniformly from the dataset and set the target subgoal to $s_{\min(t+k,T)}$. For the low-level policy (Equation (7)), we first sample a state s_t from $\mathcal{D}$, and set the input subgoal to $s_{\min(t+k,T)}$ in the same trajectory.

Advantage estimates. In principle, the advantage estimates for Equations (6) and (7) are respectively given as

$$A^h(s_t, s_{t+\tilde{k}}, g) = \gamma^{\tilde{k}} V_{\theta_V}(s_{t+\tilde{k}}, g) + \sum_{t'=t}^{\tilde{k}-1} r(s_{t'}, g) - V_{\theta_V}(s_t, g), \tag{8}$$

$$A^\ell(s_t, a_t, \tilde{s}_{t+k}) = \gamma V_{\theta_V}(s_{t+1}, \tilde{s}_{t+k}) + r(s_t, \tilde{s}_{t+k}) - V_{\theta_V}(s_t, \tilde{s}_{t+k}), \tag{9}$$

where we use the notations $\tilde{k}$ and $\tilde{s}_{t+k}$ to incorporate the edge cases discussed in the previous paragraph (*i.e.*, $\tilde{k} = \min(k, t_g - t)$ when we sample g from future states, $\tilde{k} = \min(k, T - t)$ when we sample g from random states, and $\tilde{s}_{t+k} = s_{\min(t+k,T)}$). Here, we note that $s_{t'} \neq g$ and $s_t \neq \tilde{s}_{t+k}$ always hold except for those edge cases. Thus, the reward terms in Equations (8) and (9) are mostly constants (under our reward function $r(s, g) = 0$ (if $s = g$), -1 (otherwise)), as are the third terms (with respect to the policy inputs). As such, we practically ignore these terms for simplicity, and this simplification further enables us to subsume the discount factors in the first terms into the temperature hyperparameter β. We hence use the following simplified advantage estimates, which we empirically found to lead to almost identical performances in our experiments:

$$\tilde{A}^h(s_t, s_{t+\tilde{k}}, g) = V_{\theta_V}(s_{t+\tilde{k}}, g) - V_{\theta_V}(s_t, g), \tag{10}$$

$$\tilde{A}^\ell(s_t, a_t, \tilde{s}_{t+k}) = V_{\theta_V}(s_{t+1}, \tilde{s}_{t+k}) - V_{\theta_V}(s_t, \tilde{s}_{t+k}). \tag{11}$$

State representations. We model the output of the representation function $\phi(g)$ in $V(s, \phi(g))$ with a 10-dimensional latent vector and normalize the outputs of $\phi(g)$ [53]. Empirically, we found that concatenating s to the input (*i.e.*, using $\phi([g, s])$ instead of $\phi(g)$, Figure 9), similarly to Hong et al. [39], improves performance in our experiments. While this might lose the sufficiency property of the representations (*i.e.*, Proposition 5.1), we found that the representations obtained in this way generally lead to better performance in practice, indicating that they still mostly preserve the goal information for control. We believe this is due to the imposed bottleneck on ϕ by constraining its effective dimensionality to 9 (by using normalized 10-dimensional vectors), which enforces ϕ to retain bits regarding g and to reference s only when necessary. Additionally, in pixel-based environments, we found that allowing gradient flows from the low-level policy loss (Equation (7)) to ϕ further improves performance. We ablate these choices and report the results in Appendix C.

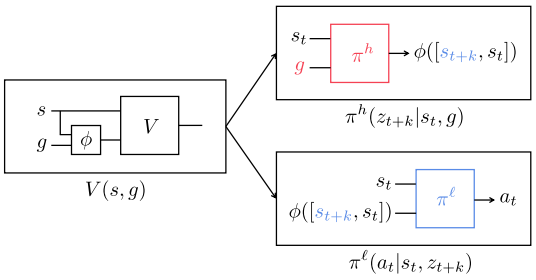

Figure 9: **Full architecture of HIQL.** In practice, we use $V(s, \phi([g, s]))$ instead of $V(s, \phi(g))$ as we found that the former leads to better empirical performance.

Policy execution. At test time, we query both the high-level and low-level policies at every step, without temporal abstraction. We found that fixing subgoal states for more than one step does not significantly affect performance, so we do not use temporal abstraction for simplicity.

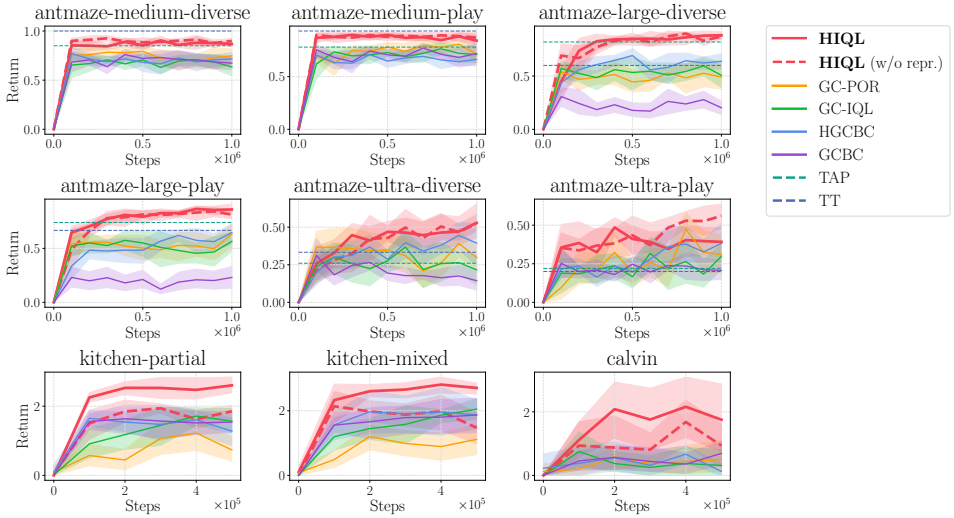

Figure 10: Training curves for the results with state-based environments (Table 1). Shaded regions denote the 95% confidence intervals across 8 random seeds.

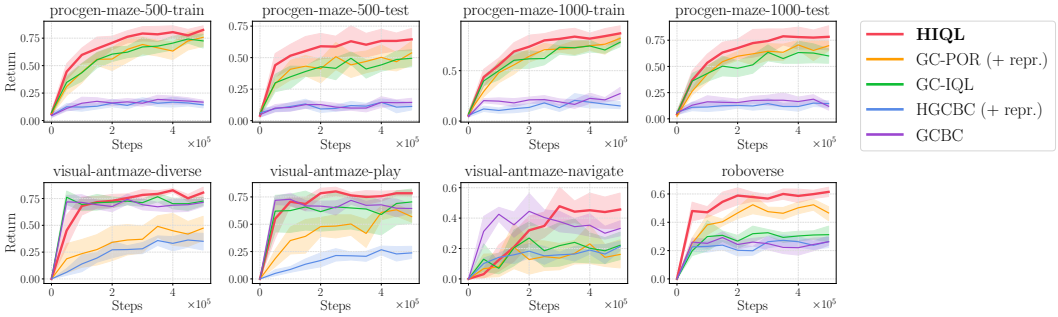

Figure 11: Training curves for the results with pixel-based environments (Table 2). Shaded regions denote the 95% confidence intervals across 8 random seeds.

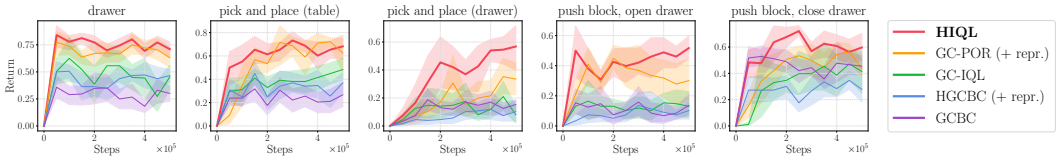

Figure 12: Training curves for the five tasks [104] in Roboverse. Shaded regions denote the 95% confidence intervals across 8 random seeds.

We provide a pseudocode for HIQL in Algorithm 1. We note that the high- and low-level policies can be jointly trained with the value function as well, as in Kostrikov et al. [49].

B Additional Plots

We include the training curves for Tables 1 to 3 in Figures 10, 11 and 13, respectively. We also provide the training curves for each of the five tasks [104] in Roboverse in Figure 12. We include the Rliable [1] plots in Figures 14 and 15. We note that the numbers in Tables 1 to 3 are *normalized* scores (see Appendix D), while the returns in the figures are unnormalized ones.

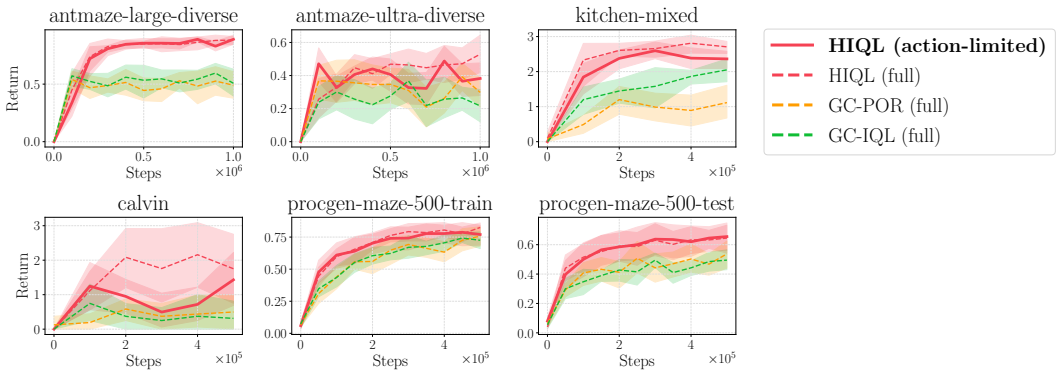

Figure 13: Training curves for the results with action-free data (Table 3). Shaded regions denote the 95% confidence intervals across 8 random seeds.

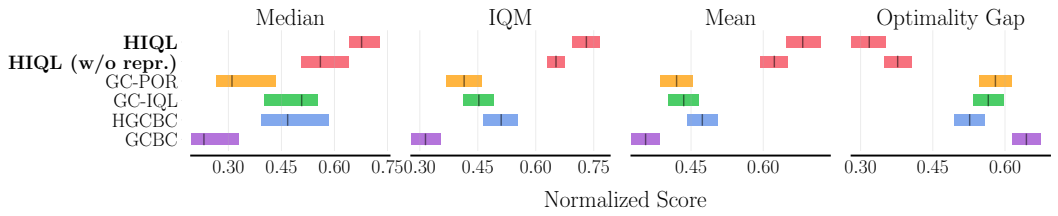

Figure 14: Rliable plots for state-based environments.

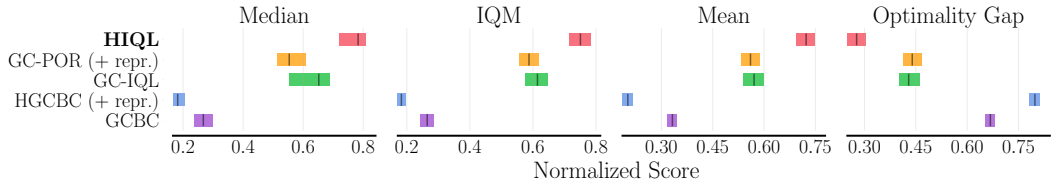

Figure 15: Rliable plots for pixel-based environments.

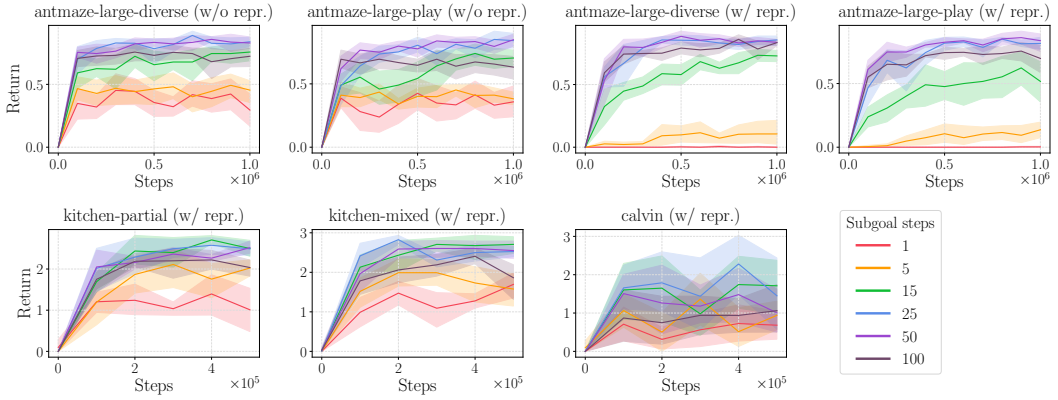

Figure 16: Ablation study of the subgoal steps k. HIQL generally achieves the best performances when k is between 25 and 50. Even when k is not within this range, HIQL mostly maintains reasonably good performance unless k is too small (*i.e.*, $\leq$ 5). Shaded regions denote the 95% confidence intervals across 8 random seeds.

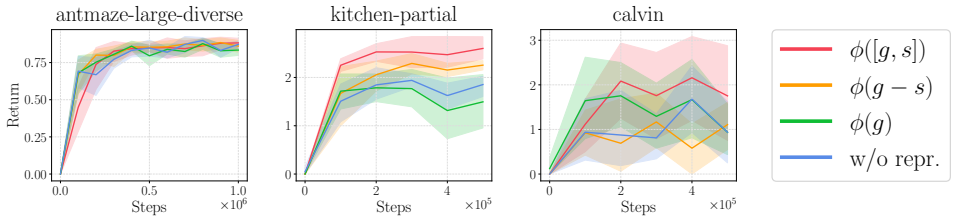

Figure 17: Ablation study of different parameterizations of the representation function. Passing s and g together to ϕ improves performance in general. Shaded regions denote the 95% confidence intervals across 8 random seeds.

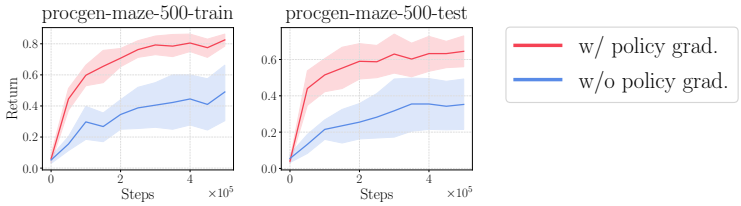

Figure 18: Ablation study of the auxiliary gradient flow from the low-level policy loss to ϕ on pixel-based ProcGen Maze. This auxiliary gradient flow helps maintain goal information in the representations. Shaded regions denote the 95% confidence intervals across 8 random seeds.

C Ablation Study

Subgoal steps. To understand how the subgoal steps k affect performance, we evaluate HIQL with six different $k \in \{1, 5, 15, 25, 50, 100\}$ on AntMaze, Kitchen, and CALVIN. On AntMaze, we test both HIQL with and without representations (Section 5.2). Figure 16 shows the results, suggesting that HIQL generally achieves the best performance with k between 25 and 50. Also, HIQL still maintains reasonable performance even when k is not within this optimal range, unless k is too small.

Representation parameterizations. We evaluate four different choices of the representation function ϕ in HIQL: $\phi([g, s])$, $\phi(g - s)$, $\phi(g)$, and without ϕ. Figure 17 shows the results, indicating that passing g and s together to ϕ generally improves performance. We hypothesize that this is because ϕ, when given both g and s, can capture contextualized information about the goals (or subgoals) with respect to the current state, which is often easier to deal with for the low-level policy. For example, in AntMaze, the agent only needs to know the relative position of the subgoal with respect to the current position.

Auxiliary gradient flows for representations. We found that in pixel-based environments (*e.g.*, Procgen Maze), allowing gradient flows from the low-level policy loss to the representation function improves performance (Figure 18). We believe this is because the additional gradients from the policy loss further help maintain the information necessary for control. We also (informally) found that this additional gradient flow occasionally slightly improves performances in the other environments as well, but we do not enable this feature in state-based environments to keep our method as simple as possible.

D Implementation details

We implement HIQL based on JaxRL Minimal [32]. Our implementation is available at the following repository: https://github.com/seohongpark/HIQL. We run our experiments on an internal GPU cluster composed of TITAN RTX and A5000 GPUs. Each experiment on state-based environments takes no more than 8 hours and each experiment on pixel-based environments takes no more than 16 hours.

D.1 Environments

AntMaze [9, 87] We use the 'antmaze-medium-diverse-v2', 'antmaze-medium-play-v2', 'antmaze-large-diverse-v2', and 'antmaze-large-play-v2' datasets from the D4RL benchmark [28]. For AntMaze-Ultra, we use the 'antmaze-ultra-diverse-v0' and 'antmaze-ultra-play-v0' datasets proposed by Jiang et al. [43]. The maze in the AntMaze-Ultra task is twice the size of the largest maze in the original D4RL dataset. Each dataset consists of 999 length-1000 trajectories, in which the Ant agent navigates from an arbitrary start location to another goal location, which does not necessarily correspond to the target evaluation goal. At test time, to specify a goal g for the policy, we set the first two state dimensions (which correspond to the x-y coordinates) to the target goal given by the environment and the remaining proprioceptive state dimensions to those of the first observation in the dataset. At evaluation, the agent gets a reward of 1 when it reaches the goal.

Kitchen [35]. We use the 'kitchen-partial-v0' and 'kitchen-mixed-v0' datasets from the D4RL benchmark [28]. Each dataset consists of 136950 transitions with varying trajectory lengths (approximately 227 steps per trajectory on average). In the 'kitchen-partial-v0' task, the goal is to achieve the four subtasks of opening the microwave, moving the kettle, turning on the light switch, and sliding the cabinet door. The dataset contains a small number of successful trajectories that achieve the four subtasks. In the 'kitchen-mixed-v0' task, the goal is to achieve the four subtasks of opening the microwave, moving the kettle, turning on the light switch, and turning on the bottom left burner. The dataset does not contain any successful demonstrations, only providing trajectories that achieve some subset of the four subtasks. At test time, to specify a goal g for the policy, we set the proprioceptive state dimensions to those of the first observation in the dataset and the other dimensions to the target kitchen configuration given by the environment. At evaluation, the agent gets a reward of 1 whenever it achieves a subtask.

CALVIN [63]. We use the offline dataset provided by Shi et al. [84], which is based on the teleoperated demonstrations from Mees et al. [63]. The task is to achieve the four subtasks of opening the drawer, turning on the lightbulb, sliding the door to the left, and turning on the LED. The dataset consists of 1204 length-499 trajectories. In each trajectory, the agent achieves some of the 34 subtasks in an arbitrary order, which makes the dataset highly task-agnostic [84]. At test time, to specify a goal g for the policy, we set the proprioceptive state dimensions to those of the first observation in the dataset and the other dimensions to the target configuration. At evaluation, the agent gets a reward of 1 whenever it achieves a subtask.

Procgen Maze [16]. We collect an offline dataset of goal-reaching behavior on the Procgen Maze suite. For each maze level, we pre-compute the optimal goal-reaching policy using an oracle, and collect a trajectory of 1000 transitions by commanding a goal, using the goal-reaching policy to reach this goal, then commanding a new goal and repeating henceforth. The 'procgen-maze-500' dataset consists of 500000 transitions collected over the first 500 levels and 'procgen-maze-1000' consists of 1000000 transitions over the first 1000 levels. At test time, we evaluate the agent on "challenging" levels that contain at least 20 leaf goal states (*i.e.*, states that have only one adjacent state in the maze). We use 50 such levels and goals for each evaluation, where they are randomly sampled either between Level 0 and Level 499 for the "-train" tasks or between Level 5000 and Level 5499 for the "-test" tasks. The agent gets a reward of 1 when it reaches the goal.

Visual AntMaze. We convert the original state-based AntMaze environment into a pixel-based environment by providing both a $64 \times 64 \times 3$-dimensional camera image (as shown in the bottom row of Figure 6b) and 27-dimensional proprioceptive states without global coordinates. For the datasets, we use the converted versions of the 'antmaze-large-diverse-v2' and 'antmaze-large-play-v2' datasets from the D4RL benchmark [28] as well as a newly collected dataset, 'antmaze-large-navigate-v2', which consists of diverse navigation behaviors that visit multiple goal locations within an episode. The task and the evaluation scheme are the same as the original state-based AntMaze environment.

Roboverse [25, 104]. We use the same dataset and tasks used in Zheng et al. [104]. The dataset consists of 3750 length-300 trajectories,[1] out of which we use the first 3334 trajectories for training

[1]While Zheng et al. [104] separate each length-300 trajectory into four length-75 trajectories, we found that using the original length-300 trajectories improves performance in general.

(which correspond to approximately 1000000 transitions), while the remaining trajectories are used as a validation set. Each trajectory in the dataset features four random primitive behaviors, such as pushing an object or opening a drawer, starting from randomized initial object poses. At test time, we employ the same five goal-reaching tasks used in Zheng et al. [104]. We provide a precomputed goal image, and the agent gets a reward of 1 upon successfully completing the task by achieving the desired object poses.

In Tables 1 to 3, we report the normalized scores with a multiplier of 100 (AntMaze, Procgen Maze, Visual AntMaze, and Roboverse) or 25 (Kitchen and CALVIN).

D.2 Hyperparameters

We present the hyperparameters used in our experiments in Table 4, where we mostly follow the network architectures and hyperparameters used by Ghosh et al. [34]. We use layer normalization [5] for all MLP layers. For pixel-based environments, we use the Impala CNN architecture [21] to handle image inputs, mostly with 512-dimensional output features, but we use normalized 10-dimensional output features for the goal encoder of HIQL's value function to make them easily predictable by the high-level policy, as discussed in Appendix A. We do not share encoders between states and goals, or between different components. As a result, in pixel-based environments, we use a total of *five* separate CNN encoders (two for the value function, two for the high-level policy, and two for the low-level policy, but the goal encoder for the value function is the same as the goal encoder for the low-level policy (Figure 1a)). In Visual AntMaze and Roboverse, we apply a random crop [48] (with probability 0.5) to prevent overfitting, following Zheng et al. [104].

During training, we periodically evaluate the performance of the learned policy at every 100K (state-based) or 50K (pixel-based) steps, using 52 (AntMaze, Kitchen, CALVIN, and Visual AntMaze), 50 (Procgen Maze), or 110 (Roboverse, 22 per each task) rollouts[2]. At evaluation, we use $\arg\max$ actions for environments with continuous action spaces and ϵ-greedy actions with $\epsilon = 0.05$ for environments with discrete action spaces (*i.e.*, Procgen Maze). Following Zheng et al. [104], in Roboverse, we add Gaussian noise with a standard deviation of 0.15 to the $\arg\max$ actions.

To ensure fair comparisons, we use the same architecture for both HIQL and four baselines (GCBC, HGCBC, GC-IQL, and GC-POR). The discount factor γ is chosen from $\{0.99, 0.995\}$, the AWR temperature β from $\{1, 3, 10\}$, the IQL expectile τ from $\{0.7, 0.9\}$ for each method.

For HIQL, we set $(\gamma, \beta, \tau) = (0.99, 1, 0.7)$ across all environments. For GC-IQL and GC-POR, we use $(\gamma, \beta, \tau) = (0.99, 3, 0.9)$ (AntMaze-Medium, AntMaze-Large, and Visual AntMaze), $(\gamma, \beta, \tau) = (0.995, 1, 0.7)$ (AntMaze-Ultra), or $(\gamma, \beta, \tau) = (0.99, 1, 0.7)$ (others). For the subgoal steps k in HIQL, we use $k = 50$ (AntMaze-Ultra), $k = 3$ (Procgen Maze and Roboverse), or $k = 25$ (others). HGCBC uses the same subgoal steps as HIQL for each environment, with the exception of AntMaze-Ultra, where we find it performs slightly better with $k = 25$. For HIQL, GC-IQL, and GC-POR, in state-based environments and Roboverse, we sample goals for high-level or flat policies from either the future states in the same trajectory (with probability 0.7) or the random states in the dataset (with probability 0.3). We sample high-level goals only from the future states in the other environments (Procgen Maze and Visual AntMaze).

E Proofs

E.1 Proof of Proposition 4.1

For simplicity, we assume that T/k is an integer and $k \leq T$.

[2]These numbers include two additional rollouts for video logging (except for Procgen Maze).

Table 4: Hyperparameters.

Hyperparameter	Value
# gradient steps	1000000 (AntMaze), 500000 (others)
Batch size	1024 (state-based), 256 (pixel-based)
Policy MLP dimensions	$(256, 256)$
Value MLP dimensions	$(512, 512, 512)$
Representation MLP dimensions (state-based)	$(512, 512, 512)$
Representation architecture (pixel-based)	Impala CNN [21]
Nonlinearity	GELU [37]
Optimizer	Adam [47]
Learning rate	0.0003
Target network smoothing coefficient	0.005

Proof. Defining $z_1 := z_{1,T}$ and $z_2 := z_{-1,T}$, the probability of the flat policy π selecting an incorrect action can be computed as follows:

$$\begin{aligned}
\mathcal{E}(\pi) &= \mathbb{P}[\hat{V}(s+1, g) \leq \hat{V}(s-1, g)] && (12)\\
&= \mathbb{P}[\hat{V}(1, T) \leq \hat{V}(-1, T)] && (13)\\
&= \mathbb{P}[-(T-1)(1+\sigma z_1) \leq -(T+1)(1+\sigma z_2)] && (14)\\
&= \mathbb{P}[z_1\sigma(T-1) - z_2\sigma(T+1) \leq -2] && (15)\\
&= \mathbb{P}[z\sigma\sqrt{T^2+1} \leq -\sqrt{2}] && (16)\\
&= \Phi\left(-\frac{\sqrt{2}}{\sigma\sqrt{T^2+1}}\right), && (17)
\end{aligned}$$

where z is a standard Gaussian random variable, and we use the fact that the sum of two independent Gaussian random variables with standard deviations of σ_1 and σ_2 follows a normal distribution with a standard deviation of $\sqrt{\sigma_1^2 + \sigma_2^2}$.

Similarly, the probability of the hierarchical policy $\pi^\ell \circ \pi^h$ selecting an incorrect action is bounded using a union bound as

$$\begin{aligned}
\mathcal{E}(\pi^\ell \circ \pi^h) &\leq \mathcal{E}(\pi^h) + \mathcal{E}(\pi^\ell) && (18)\\
&= \mathbb{P}[\hat{V}(s+k, g) \leq \hat{V}(s-k, g)] + \mathbb{P}[\hat{V}(s+1, s+k) \leq \hat{V}(s-1, s+k)] && (19)\\
&= \mathbb{P}[\hat{V}(k, T) \leq \hat{V}(-k, T)] + \mathbb{P}[\hat{V}(1, k) \leq \hat{V}(-1, k)] && (20)\\
&= \Phi\left(-\frac{\sqrt{2}}{\sigma\sqrt{(T/k)^2+1}}\right) + \Phi\left(-\frac{\sqrt{2}}{\sigma\sqrt{k^2+1}}\right). && (21)
\end{aligned}$$

□

E.2 Proof of Proposition 5.1

We first formally define some notations. For $s \in \mathcal{S}, a \in \mathcal{A}, g \in \mathcal{S}$, and a representation function $\phi : \mathcal{S} \to \mathcal{Z}$, we denote the goal-conditioned state-value function as $V(s, g)$, the action-value function as $Q(s, a, g)$, the parameterized state-value function as $V_\phi(s, z)$ with $z = \phi(g)$, and the parameterized action-value function as $Q_\phi(s, a, z)$. We assume that the environment dynamics are deterministic, and denote the deterministic transition kernel as $p(s, a) = s'$. Accordingly, we have $Q(s, a, g) = V(p(s, a), g) = V(s', g)$ and $Q_\phi(s, a, z) = V_\phi(p(s, a), z) = V_\phi(s', z)$. We denote the optimal value functions with the superscript "$*$", *e.g.*, $V^*(s, g)$. We assume that there exists a parameterized value function, which we denote $V_\phi^*(s, \phi(g))$, that is the same as the true optimal value function, *i.e.*, $V^*(s, g) = V_\phi^*(s, \phi(g))$ for all $s \in \mathcal{S}$ and $g \in \mathcal{S}$.

Proof. For π^*, we have

$$\pi^*(a \mid s, g) = \arg\max_{a \in \mathcal{A}} Q^*(s, a, g) \tag{22}$$

$$= \arg\max_{s' \in \mathcal{N}_s} V^*(s', g) \tag{23}$$

$$= \arg\max_{s' \in \mathcal{N}_s} V^*_\phi(s', z), \tag{24}$$

where $\mathcal{N}_s$ denotes the neighborhood sets of s, *i.e.*, $\mathcal{N}_s = \{s' \mid \exists a, p(s, a) = s'\}$. For π^*_ϕ, we have

$$\pi^*_\phi(a \mid s, z) = \arg\max_{a \in \mathcal{A}} Q^*_\phi(s, a, z) \tag{25}$$

$$= \arg\max_{s' \in \mathcal{N}_s} V^*_\phi(s', z). \tag{26}$$

By comparing Equation (24) and Equation (26), we can see that they have the same $\arg\max$ action sets for all s and g. □

F Subgoal Visualizations

We visualize learned subgoals in Figures 19 and 20 (videos are available at https://seohong.me/projects/hiql/). For AntMaze-Large, we train HIQL without representations and plot the x-y coordinates of subgoals. For Procgen Maze, we train HIQL with 10-dimensional representations and find the maze positions that have the closest representations (with respect to the Euclidean distance) to the subgoals produced by the high-level policy. The results show that HIQL learns appropriate k-step subgoals that lead to the target goal.

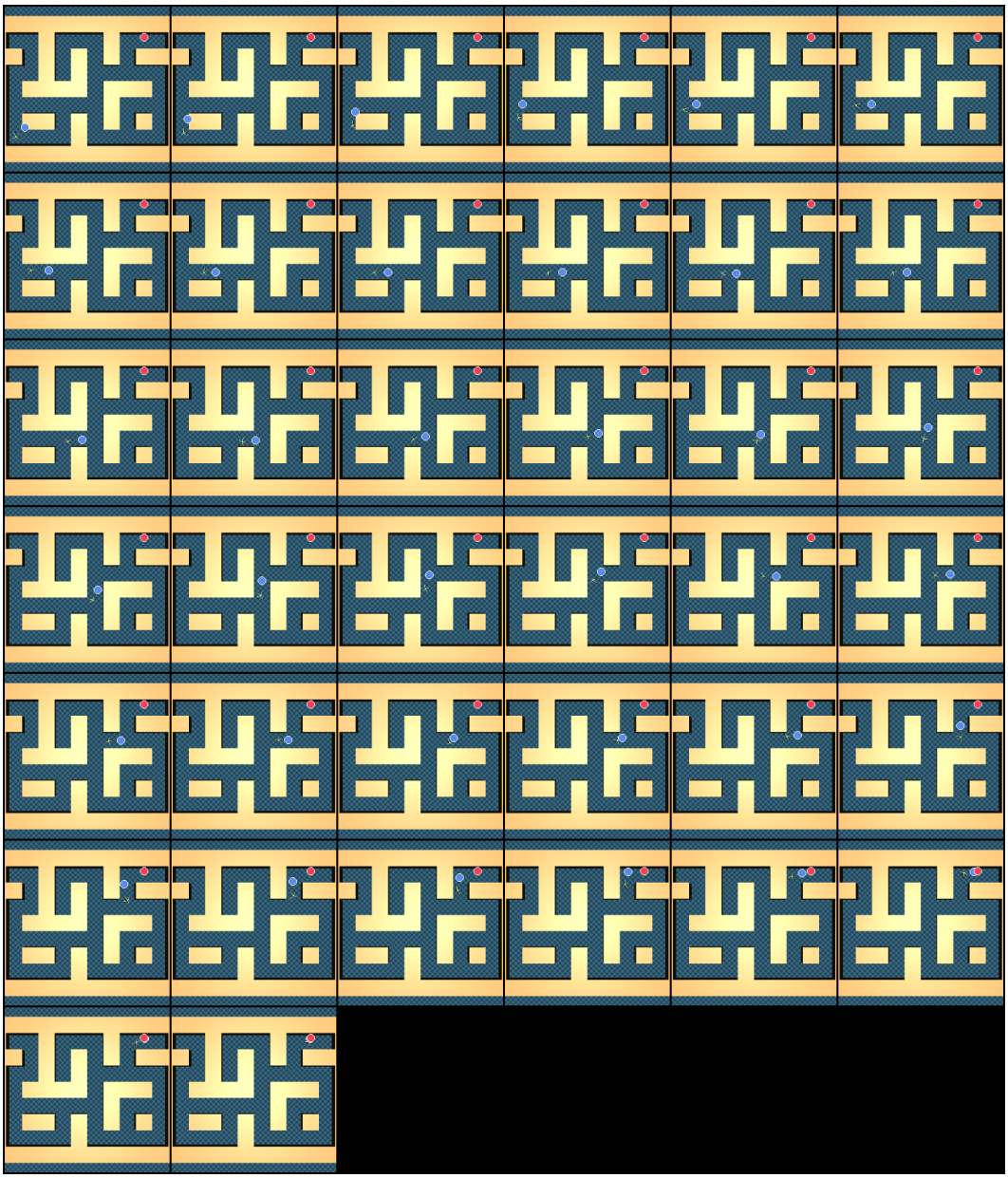

Figure 19: Subgoal visualization in AntMaze-Large. The red circles denote the target goal and the blue circles denote the learned subgoals. Videos are available at https://seohong.me/projects/hiql/.

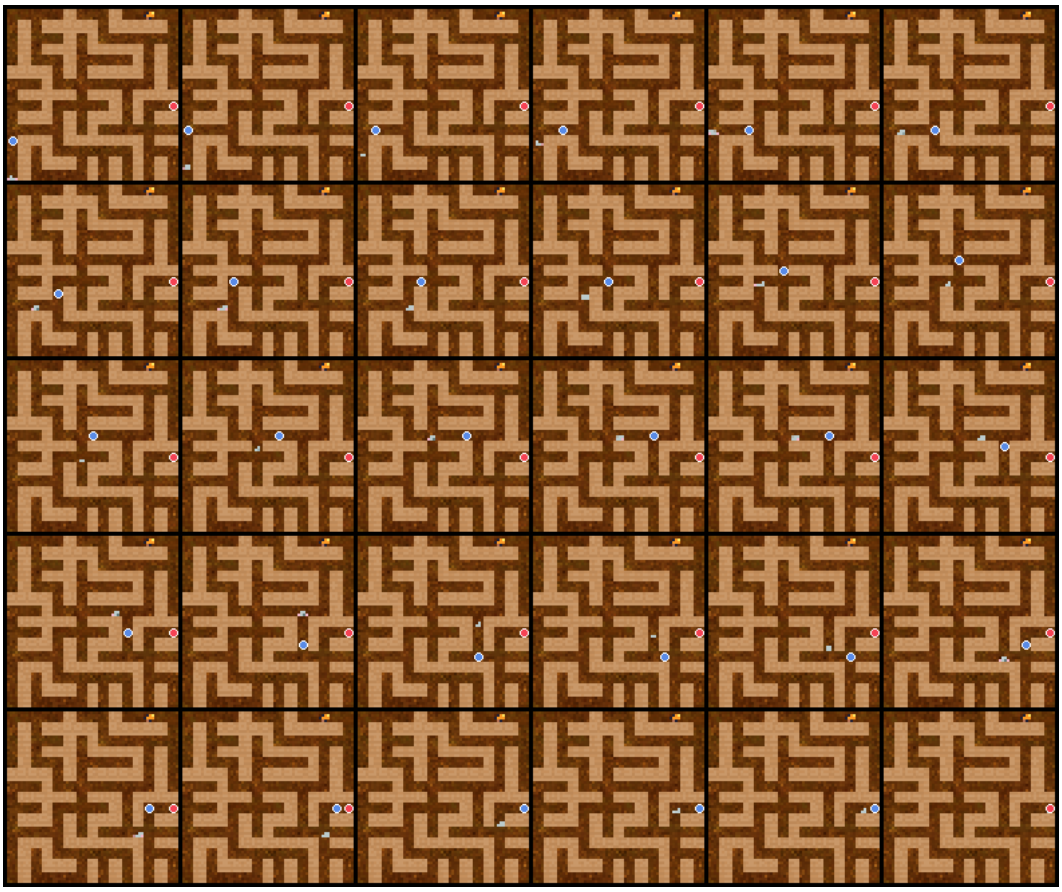

Figure 20: Subgoal visualization in Procgen Maze. The red circles denote the target goal, the blue circles denote the learned subgoals, and the white blobs denote the agent. Videos are available at https://seohong.me/projects/hiql/.