A. High-level Policy Derivation

The following derivation is similar to (Abdolmaleki et al., 2018b) and (Nair et al., 2020). We consider the subgoal advantage function A^{π^H}, some distribution over valid states $p_s(s)$ and the distribution over states and goals $\mu(s,g)$ given by previously collected experience and experience relabelling. During high-level policy improvement, we maximize the following objective function, where expectations have been replaced by integrals:

$$\begin{aligned} \pi^H_{k+1} &= \arg\max_{\pi} \iint \mu(s,g) \int \pi^H(s_g|s,g) A^{\pi^H}(s_g|s,g) ds_g ds dg \\ \text{s.t.} \iint \mu(s,g) D_{\mathrm{KL}}\left(\pi^H(.|s,g) \,||\, p_s(.)\right) ds dg &\leq \epsilon, \iint \mu(s,g) \int \pi^H(s_g|s,g) ds_g ds dg = 1 \end{aligned} \tag{11}$$

The Lagrangian is:

$$\begin{aligned} \mathcal{L}(\pi^H, \lambda, \eta) = &\iint \mu(s,g) \int \pi^H(s_g|s,g) A^{\pi^H}(s_g|s,g) ds_g ds dg \\ &+ \lambda \left(\epsilon - \iint \mu(s,g) \int \pi^H(s_g|s,g) \log \frac{\pi^H(s_g|s,g)}{p_s(s_g)} ds_g ds dg \right) \\ &+ \eta \left(1 - \iint \mu(s,g) \int \pi^H(s_g|s,g) ds_g ds dg \right). \end{aligned} \tag{12}$$

Differentiating with respect to π^H yields

$$\frac{\partial \mathcal{L}}{\partial \pi^H} = A^{\pi^H}(s_g|s,g) - \lambda \log \pi^H(s_g|s,g) + \lambda \log p_s(s_g) + \eta - \lambda. \tag{13}$$

Setting this expression to zero, we get

$$\pi^H(s_g|s,g) = p_s(s_g) \exp\left(\frac{A^{\pi^H}(s_g|s,g)}{\lambda} \right) \exp\left(-\frac{\lambda - \eta}{\lambda} \right). \tag{14}$$

As the second constraint in (11) must sum to 1, the last exponential is a normalizing factor. This gives the closed form solution:

$$\pi^H_\star(s_g|s,g) = \frac{1}{Z(s,g)} p_s(s_g) \exp\left(\frac{A^{\pi^H}(s_g|s,g)}{\lambda} \right) \tag{15}$$

with the normalizing partition function

$$Z(s,g) = \int p_s(s_g) \exp\left(\frac{1}{\lambda} A^{\pi^H}(s_g|s,g) \right) ds_g. \tag{16}$$

We next project the closed form solution into the space of parametric policies by minimizing the reverse KL divergence between our parametric high-level policy π^H_ψ and the optimal non-parametric solution $\pi^H_\star$:

$$\begin{aligned} \pi^H_{\psi_{k+1}} &= \arg\min_{\psi} \mathbb{E}_{(s,g)\sim\mu(.)} \left[D_{\mathrm{KL}}\left(\pi^H_\star(.|s,g) \,||\, \pi^H_\psi(.|s,g) \right) \right] \\ &= \arg\max_{\psi} \mathbb{E}_{(s,g)\sim\mu(.)} \mathbb{E}_{s_g\sim\pi^H_\star(.|s,g)} \left[\log \pi^H_\psi(s_g|s,g) \right] \\ &= \arg\max_{\psi} \mathbb{E}_{(s,g)\sim\mu(.), s_g\sim p_s(.)} \left[\log \pi^H_\psi(s_g|s,g) \frac{1}{Z(s,g)} \exp\left(\frac{1}{\lambda} A^{\pi^H}(s_g|s,g) \right) \right] \end{aligned} \tag{17}$$

We choose p_s to be the distribution of states given by experience collected in the replay buffer D, such that high-level policy improvement corresponds to a weighted maximum-likelihood where subgoal candidates s_g are randomly sampled from D among states visited by the agent in previous episodes.

B. Implementation Details

Actor-Critic. Our implementation of the actor-critic algorithm is based on Soft Actor-Critic (Haarnoja et al., 2018b), where we remove the entropy term during policy evaluation and replace the entropy term by the KL divergence between policy and our prior policy during policy improvement. The policy is a neural network that parametrizes the mean and diagonal covariance matrix of a squashed Gaussian distribution $\pi_\theta(.|s,g) = \tanh \mathcal{N}(\mu_\theta(s,g), \Sigma_\theta(s,g))$. We train two seperate Q-networks with target networks and take the minimum over the two target values to compute the bootstrap value. The target networks are updated using an exponential moving average of the online Q parameters: $\phi'_{k+1} = \tau\phi_k + (1-\tau)\phi'_k$.

High-level policy training. The high-level policy is a neural network that outputs the mean and diagonal covariance matrix of a Laplace distribution $\pi^H_\psi(.|s,g) = \text{Laplace}(\mu_\psi(s,g), \Sigma_\psi(s,g))$. Following (Nair et al., 2020), instead of estimating the normalizing factor $Z(s,g)$ in (17), we found that computing the weights as softmax of the advantages over the batch leads to good results in practice. During the high-level policy improvement, we found that clipping the value function between -100 and 0, which corresponds to the expected bounds given our choice of reward function and discount factor, stabilizes the training slightly.

KL divergence estimation. We use an exponentially moving average of the policy weights instead of the weights of the current policy to construct the prior policy π_{prior}: $\theta'_{k+1} = \tau\theta_k + (1-\tau)\theta'_k$ with the same smoothing coefficient τ as the one used for the Q function. We estimate the prior density using the following Monte-Carlo estimate:

$$\log \pi^{prior}(a|s,g) \approx \log \left[\frac{1}{I} \sum_i \pi_{\theta'}(a|s, s^i_g) + \epsilon \right], (s^i_g) \sim \pi^H_\psi(.|s,g), \tag{18}$$

where $\epsilon > 0$ is a small constant to avoid large negative values of the prior log-density. We use $I = 10$ samples to estimate $\pi^{prior}(a|s,g)$. We also use a Monte-Carlo approximation to estimate the KL-divergence term in Equation (9) of the submission:

$$\begin{aligned} D_{\text{KL}}\left(\pi_\theta(.|s,g) \,||\, \pi^{prior}(.|s,g)\right) &= \mathbb{E}_{a\sim\pi(.|s,g)}[\log \pi_\theta(a|s,g) - \log \pi^{prior}(a|s,g)] \\ &\approx \frac{1}{N} \sum_n \left[\log \pi_\theta(a_n|s,g) - \log \pi^{prior}(a_n|s,g)\right] \\ \text{with } (a_n)_{n=1,..,N} &\sim \pi_\theta(.|s,g). \end{aligned} \tag{19}$$

Following SAC (Haarnoja et al., 2018a), we use $N = 1$, plug the estimate (18) and use the reparametrization trick to backpropagate the KL divergence term to the policy weights (Haarnoja et al., 2018a).

Experience relabelling. In all of our experiments we use Hindsight Experience Replay (Andrychowicz et al., 2017). We use the same relabelling strategy as (Nair et al., 2018) and (Nasiriany et al., 2019) and relabel the goals in our minibatches as follows:

- 20%: original goals from collected trajectories,
- 40%: randomly sampled states from the replay buffer, trajectories,
- 40%: future states along the same collected trajectory.

Vision based environments On the vision-based robotic manipulation tasks, input images are passed through an image encoder shared between the policy, high-level policy and Q-function. Both states, goals and subgoals are encoded using the same encoder network. The encoder is updated during policy evaluation, where we only update the representation of the current state images whereas the representations of desired goal images, next state images and subgoal image candidates are kept fixed. We augment the observations with random translations by translating the 84×84 image within a 100×100 empty frame (Laskin et al., 2020; Kostrikov et al., 2020).

C. Environments

We adopt Ant Navigation environments provided by the code repository of (Nasiriany et al., 2019). In these environments, a 4-legged ant robot must learn to navigate in various mazes. The state includes the position, orientation, the joint angles

and the velocities of these components. The ant has a radius of roughly 0.75 units. We consider that the goal is reached if the x-y position of the ant is within 0.5 units of its desired location in Euclidean distance. The agent receives a reward $r(s, a, g) = -1$ for all actions until the goal is reached. The dimensions of the space for the U-shaped maze are 7.5×18 units. For the S-shaped maze, the dimensions are 12×12. For the Π-shaped and ω-shaped mazes, the dimensions are 16×16 units. The walls are 1.5 units thick. During training, initial states and goals are uniformly sampled anywhere in the empty space of the environment. At test time, we evaluate the agent on challenging configurations that require temporally extended reasoning, as illustrated in Figure 6 of the submission.

For the robotic manipulation task, we use the same image-based environment as in (Nasiriany et al., 2019). In this task, the agent operates an arm robot via 2D position control and must manipulate a puck. The agent observes a 84×84 RGB image showing a top-down view of the scene. The dimension of the workspace are 40cm $\times$ 20cm and the puck has a radius of 4cm. We consider that the goal is achieved if both the arm and the puck are within 5cm of their respective target positions. During training, the initial arm and puck positions and their respective desired positions are uniformly sampled in the workspace. At test time, we evaluate the policy on a hard configuration which requires temporally extended reasoning: the robot must reach across the table to a corner where the puck is located, move its arm around the puck, pull the puck to a different corner of the table, and reach again the opposite corner.

D. Hyperparameters

Table 1 lists the hyperparameters used for the RIS. We use Adam optimizer and report results after one million interactions with the environment. For SAC, following (Haarnoja et al., 2018b), we automatically tune the entropy of the policy to match the target entropy of $-\dim(\mathcal{A})$.

Table 1. Hyper-parameters for RIS and SAC.

Hyper-parameter	Ant Navigation	Robotic Manipulation
Q hidden sizes	$[256, 256]$	$[256, 256]$
Policy hidden sizes	$[256, 256]$	$[256, 256]$
High-level policy hidden sizes	$[256, 256]$	$[256, 256]$
Hidden activation functions	ReLU	ReLU
Batch size	2048	256
Training batches per environment step	1	1
Replay buffer size	1×10^6	1×10^5
Discount factor γ	0.99	0.99
polyak for target networks τ	5×10^{-3}	5×10^{-3}
ϵ	1×10^{-16}	1×10^{-4}
Critic learning rate	1×10^{-3}	1×10^{-3}
Policy learning rates	1×10^{-3}	1×10^{-4}
High-level policy learning rate	1×10^{-4}	1×10^{-4}
α	0.1	0.1
λ	0.1	0.1

In the vision based environment, our image encoder is a serie of convolutional layers with kernel sizes $[3, 3, 3, 3]$, strides $[2, 2, 2, 1]$, channel sizes $[32, 32, 32, 32]$ and $ReLU$ activation functions followed by a fully-connected layer with output dimension 16.

Table 2. Environment specific hyper-parameters for LEAP

Hyperparameter	U-shaped maze	S-shaped maze	Π-shaped maze	ω-shaped Maze	Robotic Manipulation
TDM policy horizon	50	50	75	100	25
Number of subgoals	11	11	11	11	3

For LEAP, we re-implemented (Nasiriany et al., 2019) and train TDM (Pong et al., 2018) policies and Q networks with hidden layers of size $[400, 300]$ and $ReLU$ activation functions. In the ant navigation environments, we pretrain VAEs with mean squared reconstruction error loss and hidden layers of size $[64, 128, 64]$, $ReLU$ activation functions and representation

size of 8 for the encoders and the decoders. In the vision based robotic manipulation environment, we pretrain VAEs with mean squared error reconstruction loss and convolutional layers with encoder kernel of sizes $[5, 5, 5]$, encoder strides of sizes $[3, 3, 3]$, encoder channels of sizes $[16, 16, 32]$, decoder kernel sizes of sizes $[5, 6, 6]$, decoder strides of sizes $[3, 3, 3]$, and decoder channels of sizes $[32, 32, 16]$, representation size of 16 and $ReLU$ activation functions. Table 2 reports the policy horizon used for each environment as well as the number of subgoals in the test configuration for the results in Figure 6 of the submission. For the results presented in Figure 7 of the submission, we adapted the number of subgoals according to the difficulty of each configuration.

E. Additional Results

In Figure 10, we provide image reconstructions of imagined subgoals on additional configurations of the vision-based robotic manipulation environment.

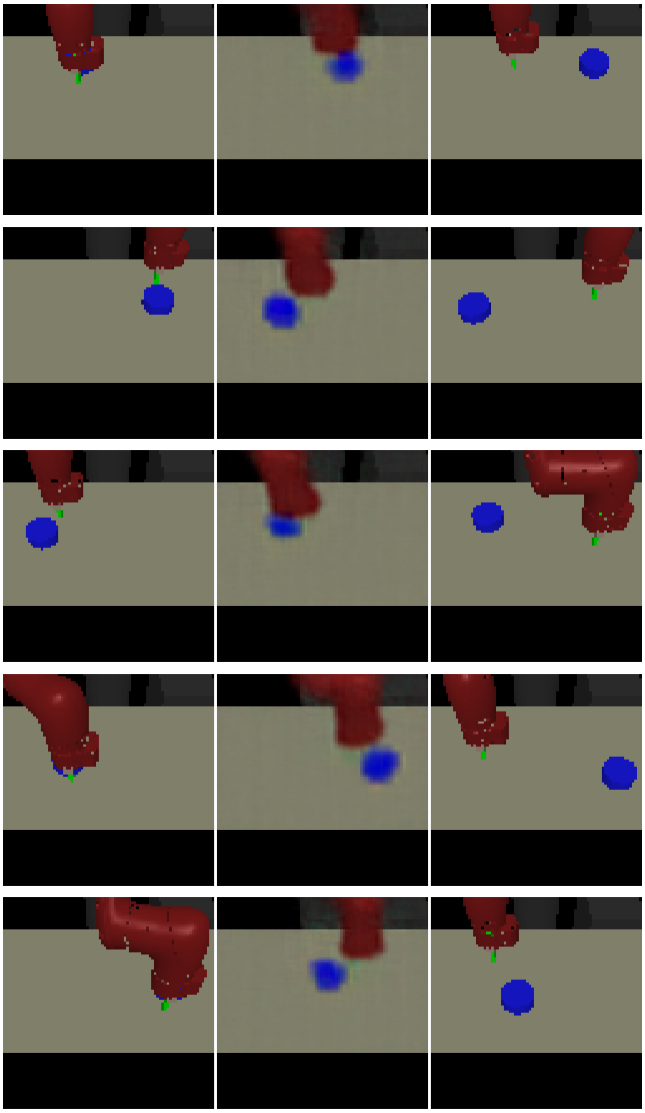

Figure 10. Image reconstruction of imagined subgoals (middle column) given current states (left column) and desired goals (right column) for different random configurations in the vision-based robotic manipulation environment.