Appendix

A Teleoperation Interface

A.1 Simulation

To collect data for simulation the human teleoperator uses the HTC VIVE Pro headset to visualize the CALVIN environment and its controller to collect play data. Clicking the application menu button emits a signal to start recording a play data episode, since then the robot arm tracks the controller's position and orientation and maps it to the gripper end effector position and orientation.

Control	Function
Application menu	Starts or finishes recording a play data episode
Trigger	When pressed it closes the gripper end effector, otherwise it opens

A.2 Real World

For the real world we use the HTC VIVE Pro controller. At the beginning of the teleoperation, we calibrate our X axis position by clicking the grip button and moving towards its positive axis, we then click the application menu to indicate that we finished the calibration movement. This calibration procedure allows us to define the reference frame in which the global coordinates from the robot end effector is be recorded. In this setting, the teleoperator is facing the table in the same direction as the robot, hence if the controller moves right so does the robot. This way we can map the position of the controller to an absolute action defining the desired robot arm end effector 3D position. Additionally, we must press continuously the grip button to enable movement of the robot arm, this is to avoid involuntary hand tracking and prevent possible accidents.

Control	Function
Grip	Starts calibration; enables robot movement
Application menu	Finishes calibration; Starts or finishes recording a play data episode
Trigger	When pressed it closes the gripper end effector, otherwise it opens

B Experimental Setup Details

For both our real and simulated robot environments we use the following 7-dimensional action scheme:

$$[\delta x, \delta y, \delta z, \delta alpha, \delta beta, \delta gamma, gripperAction]$$

The $\delta x, \delta y, \delta z$ action dimensions corresponds to a change in the end effector's position in 3D space. The $\delta alpha, \delta beta, \delta gamma$ action dimensions specifies a change in the end-effector's orientation in the robot's base frame. All these 6 dimensions accept continuous values between $[-1, 1]$. Finally, the $gripperAction$ command can have two discrete values -1.0 and 1.0. An action of -1.0 in this dimension commands the robot to open the gripper and 1.0 indicates that we desire to close it.

B.1 Data Collection Details

For the real world robot, we collected nine hours of play data by teleoperating a Franka Emika Panda robot arm, were we also manipulate objects in a 3D tabletop environment. This environment consists of a table with a drawer that can be opened and closed. The environment also contains a sliding door on top of a wooden base, such that the handle can be reached by the end effector. On top of the drawer, there are three led buttons with green, blue and orange coatings to be able to identify them, on the recorded play data we only interacted with the led button with green coating. When the led button is clicked, it toggles the state of the light. Additionally, there are three different colored blocks with letters on top. We visualize the data collection and setup in Figure 5.

In every time step we record the measurements from the robot proprioceptive sensors, as well as an static RGB-D image of size 150×200 from the Azure Kinect camera and a gripper RGB-D image of size 200×200 emitted by the FRAMOS Industrial Depth Camera D435e and the commanded absolute action. For this project we only use the static RGB image and the gripper camera RGB image as part of the observation. We visualize the input observations in Figure 6.
We extract the relative action a_t from the change in the absolute actions from time steps a_t and a_{t-1} as in practice, reproducing the relative actions computed this way showed a better performance than

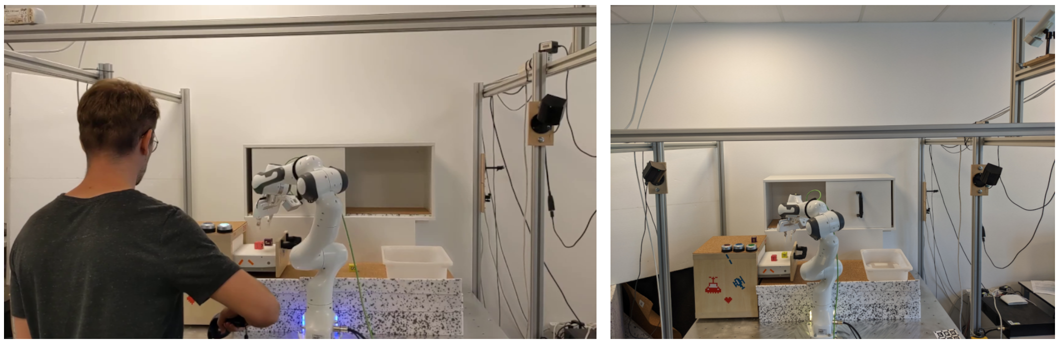
Figure 5. Visualization of the real world data collection procedure (left) and the full robot setup (right).

when computed from the noisy measurements of the proprioceptive sensors.
The data is collected at 30 Hz, but given that there is relatively a very small change in the end effector position between frames, we downsample the processed data to a frequency of 15 Hz.

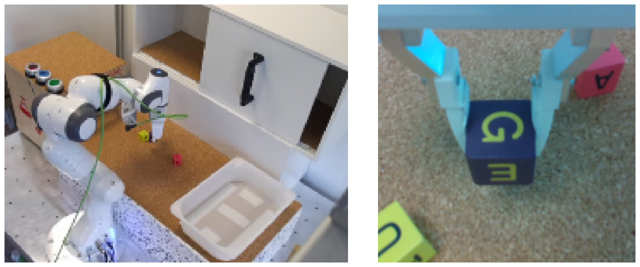
Figure 6. Visualization of the observations obtained in the real world environment. On the left we show the RGB image captured by the static camera. On the right we show the RGB image captured by the gripper camera.

C Network Architecture

C.1 Hyperparameters for TACO-RL

Low-Level Policy: To learn the low-level policy we trained the model using 8 gpus with Distributed Data Parallel. Throughout training, we randomly sample windows between length 8 and 16 and pad them until reaching the max length of 16 by repeating the last observation and an action equivalent to keeping the end effector in the same state. We visualize the low-level policy architecture in Figure 7. Additional hyperparameters are listed below:

- Batch size of 64
- Latent plan dim of 16
- Learning rate of $1e-4$
- The KL loss weight β is $1e-3$ and uses KL balancing

These hyperparameters were chosen in order to make the action space of the high-level policy tractable for reinforcement learning, as the latent plan z is used as an action when learning the high-level policy.

High-Level Policy: Similarly as in the low-level policy, we also trained the model using 8 gpus with Distributed Data Parallel. We visualize the architecture in Figure 8. We use the following hyperparameters:

- Batch size of 64
- Learning rates – Q-function: $3e-4$, Policy: $1e-4$,
- Target network update rate of 0.005
- Ratio of policy to Q-function updates of $1:1$
- Number of Q-functions: 2 Q-functions, min(Q1, Q2) used for Q-function backup and policy update

- Automatic entropy tuning: $True$, with target entropy set to $-\log|A|$
- CQL version: $CQL(\mathcal{H})$ (Using deterministic backup)
- α in CQL: 1.0 (we used the non-Lagrange version of $CQL(\mathcal{H})$)
- Number of negative samples used for estimating $logsumexp$: 4
- Initial BC warmstart epochs: 5
- Discount factor of 0.95

It is important to mention that the model performance is robust to slight changes in the hyperparameter selection.

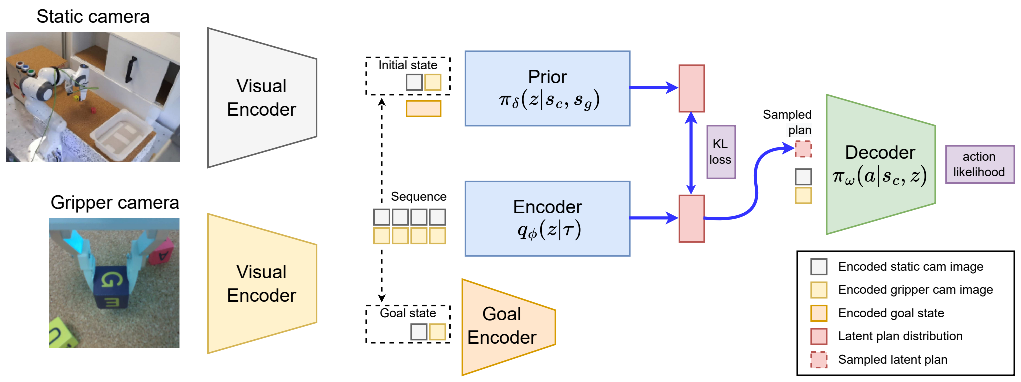

Figure 7. Overview of the architecture used in the real world to learn the low-level policy.

Goal Sampling Strategy: We use the same geometric distribution probability of $p = 0.3$ for all experiments. When this hyperparameter is closer to 0 it will be able to stitch plans to achieve longer horizon tasks, but it will encounter less often a reward of 1 which makes the optimization problem harder. For the 3D tabletop environment in both simulation and real world, we sample positive examples (goals from the future) 90% of the time and negative examples (goals with similar proprioceptive state) 10% of the time.

C.2 Encoder

The encoder (aka Posterior) $q_\phi(z|\tau)$ encodes the trajectory τ of state-action pairs into a distribution in latent space and gives out parameters of that distribution. In our case, we represent $q_\phi(z|\tau)$ with a transformer network, which takes τ and outputs parameters of a Gaussian distribution $(\mu_z^{enc}, \sigma_z^{enc})$. We encode the sequence of visual observations with our vision encoder. Then, we add positional embeddings to enable the experience window to carry temporal information. Finally, we fed the result into the transformer to learn temporally contextualized global video representations. In particular, our transformer encoder architecture consists of 2 blocks, 8 self-attention heads, and a hidden dimension of 2048.

C.3 Decoder

The decoder (aka Low-Level Policy): $\pi_\omega(a|s_c, z)$ is the latent-conditioned policy. It maximizes the conditional log-likelihood of actions in τ given the state and the latent vector. In our implementation, we parameterize it as a recurrent neural network which takes as input the current state and the sampled latent plan and gives out parameters of a discretized logistic mixture distribution [52].
For our experiments, we use 2 RNN layers each containing a hidden dimension of 2048. In the discretized logistic mixture distribution we use 10 distributions, predicting 10 classes per dimension. We predict an action where the 6 first dimensions are continuous in a range between -1.0 and 1.0 and its last dimension contains the gripper action, which is a discrete value optimized by cross entropy loss.

C.4 Prior

The prior $\pi_\delta(z|s_c, s_g)$ tries to predict the encoded distribution of the trajectory τ from its initial state and its goal state. Our implementation uses a feed-forward neural network which takes in the embedded representation of the initial state and goal state and predicts the parameters of a Gaussian distribution $(\mu_z^{pr}, \sigma_z^{pr})$. The prior network consists of 3 fully connected layers with a size of 256.

C.5 Visual Encoder

To obtain the current state embedded representation we pass each input modality observation through a convolutional encoder and we perform late fusion by concatenating the embedded representations of all the observation modalities.

Simulation

In simulation we only use the RGB static image as input, this is passed through the same convolutional encoder proposed in the original Play LMP implementation with 3 convolulational layers and a spatial softmax layer, after flattening its representation it is fed through two fully connected layers which output an embedded latent size of 32.

Real World

For the real world we use both RGB static image and the RGB gripper image as input. We send the RGB gripper image to a convolutional encoder as in original Play LMP implementation, with an embedded latent size of 32. For the RGB static image we used the pretrained R3M [53] Resnet18 networks with fixed weights, afterwards we passed it through 2 feedforward networks with a hidden size of 256 and an embedded latent size of 32.

C.6 Goal Encoder

We obtain a compact perceptual representation from the goal observation by passing it through the visual encoder. Then, we use 2 hidden layers with a size of 256 and an output layer that maps its representation to 32 latent features.

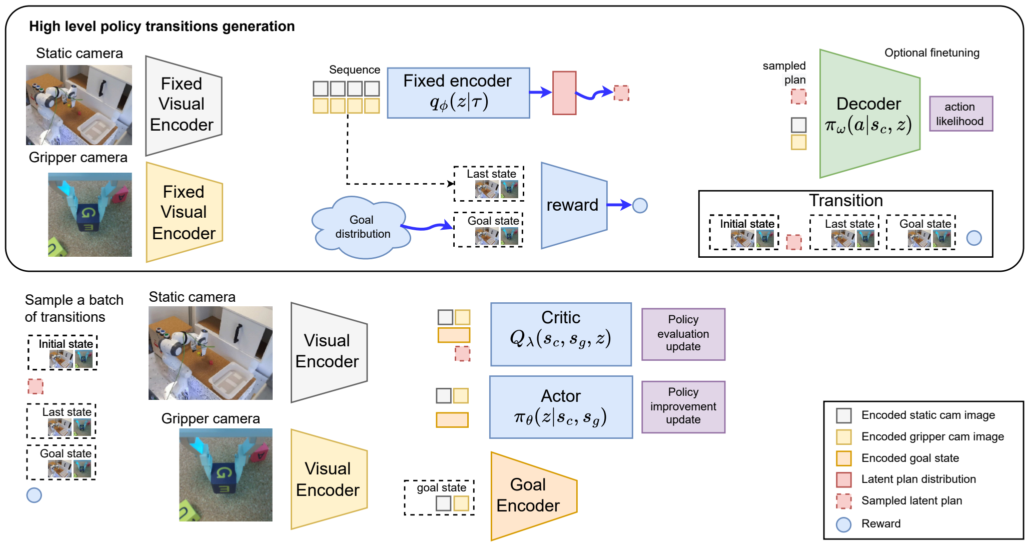

Figure 8. Overview of the architecture used in the real world to learn the high-level policy. We first used the learned low level policy to generate transitions using latent plans, afterwards we sample a batch of transitions and use CQL to learn the high level policy.

C.7 Actor

To learn the high level policy we use CQL. In particular, The actor network uses the same network architecture as the prior $\pi_\delta(z|s_c, s_g)$, as we can use the pre-learned weights as a good initialization. It has 3 hidden fully connected layers with a size of 256 and an output layer that predicts the mean and standard deviation of a latent plan Gaussian distribution.

C.8 Critic

The critic network takes as input the embedded representation of the current state, the encoded goal state and a sampled latent plan. All the inputs are concatenated and passed through a feed forward network with 3 hidden layers with a size of 256 and an output layer which predicts the expected state-action value of the policy.

D Policy Training Details

In this section we specify some implementation details of our baseline models used as comparison against TACO-RL.

D.1 Play-supervised Latent Motor Plans (LMP)

For our reimplementation of LMP we used the same network architecture described in C, the main differences with the original implementation is that the latent goal representation is only added to the prior, but not to the decoder. Additionally, we apply a $\tanh$ transformation to the sampled latent plan to ensure that every dimension is between -1.0 and 1.0. Another difference with respect to the original formulation is that we implement a KL balancing. As the KL-loss is bidirectional, we want to avoid regularizing the plans generated by the posterior toward a poorly trained prior. To solve this problem, we minimize the KL-loss faster with respect to the prior than the posterior by using different learning rates, $\alpha = 0.8$ for the prior and $1 - \alpha$ for the posterior, similar to Hafner *et al.* [48]. These architectural changes were made to do a fair comparison against our implementation, and to make sure the improvements of our method were due to the plan stitching capabilities and our self-supervised reward function.
Additionally, the decoder predicts relative actions instead of absolute actions as this leads to an overall better performance. We note that our LMP baseline is implemented in the same way.

D.2 Conservative Q-Learning with Hindsight Experience Replay (CQL + HER)

We use the same visual encoder architecture as in TACO-RL. The critic networks are made with 3 fully connected layers using a hidden dimension of 256. The actor network also uses 3 fully connected layers with a size of 256 and predicts in the last dimension a discrete action to open and close the parallel gripper. This last dimension is predicted through a Gumbel Softmax distribution.
The self-supervised reward function is done similarly as in our method with a small difference when sampling goal states from the future. The CQL transitions are (s_t, a, s_{t+1}) instead of (s_t, z_t, s_{t+k-1}), therefore when we sample a goal from discrete-time offset $\Delta \sim Geom(p)$, we use the observations at the time step $t + \Delta$ as a goal instead of the observation at the timestep $t + \Delta * (k-1)$, where k is the window size and p is the same value for both implementations. This change is required as CQL needs to take isolated decisions every time step. In contrast, our formulation allows TACO-RL to reduce the effective episode horizon by predicting high-level actions (latent plans).

D.3 Relay Imitation Learning (RIL)

We use the same visual encoder architecture as in TACO-RL to do a fair comparison. For the policy architecture, we first tried the architecture proposed by the original authors of each policy using two layer neural networks with 256 units each and ReLu nonlinearities. This architecture obtained poor performance for our application, we noticed that the respective loss objectives were not being correctly optimized for which we solved this issue by training a bigger network. For our tested implementation, for both the high-level and low-level policy we used four layer neural networks with 1024 units each.

E Data Preprocessing

E.1 Simulation

In simulation we only use the static camera RGB image as input, we first resize the RGB image from 200×200 to 128×128. Then we perform stochastic image shifts of $0 - 6$ pixels to the static camera images and a bilinear interpolation is applied on top of the shifted image by replacing each pixel with the average of the nearest pixels. Then we apply a color jitter transform augmentation with a contrast of 0.1, a brightness of 0.1 and hue of 0.02. Finally, we normalize the input image to have pixels with float values between -1.0 and 1.0.

E.2 Real World

In the real world as there are several occlusions only using the static camera RGB image, we also add the gripper camera RGB image to the observation. For the static camera RGB image we use the original size of 150×200, we then apply a color jitter transform with contrast of 0.05, a brightness of 0.05 and a hue of 0.02. Finally, we use the values for the pretrained R3M normalization, i.e., mean $= [0.485, 0.456, 0.406]$ and a standard deviation, std $= [0.229, 0.224, 0.225]$.
For the gripper camera RGB image we resize the image from 200×200 to 84×84, we then apply a color jitter transform with contrast of 0.05, a brightness of 0.05 and a hue of 0.02. Then we perform stochastic image shifts of $0 - 4$ pixels to the and a bilinear interpolation is applied on top of the shifted image by replacing each pixel with the average of the nearest pixels. Finally, we normalize the input image to have pixels with float values between -1.0 and 1.0.

F Additional Results

F.1 CALVIN

We add the rest of the results of all the different tasks that could be made in the CALVIN environment where the goal image does not need to contain the robot end effector performing the task in Figure 9. We run 50 rollouts for each task. Each rollout uses a different goal image which was not seen throughout training.

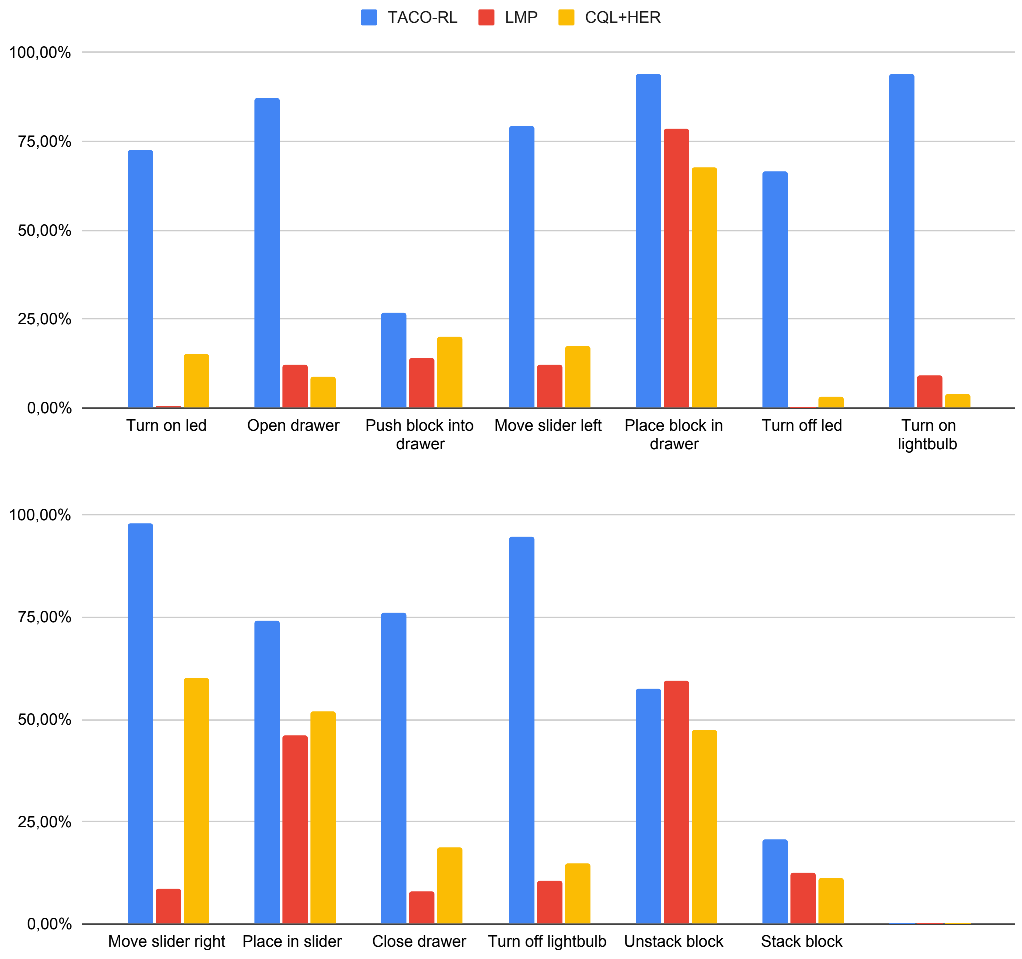

Figure 9. The average success rate of goal-conditioned models running 50 rollouts where the goal image does not contain the end effector performing the task. Results were calculated using 3 seeds.

F.2 Maze2D

We also evaluate our algorithm in the Maze2D environments from D4RL [54]. These tasks are designed to provide a simple test that the model is capable of stitching together various subtrajectories such that the agent is capable of reaching the goal in the smallest amount of steps. For TACO-RL, we learn the policy by relabeling the transitions instead of using the rewards given by the dataset. Additionally, given that the scene does not change through time, we only add positive examples in our goal sampling approach. We perform the rollouts of both LMP and TACO-RL by exposing the desired target position. TACO-RL outperforms both LMP and CQL as it is able to stitch plans through dynamic programming as shown in Table 5.

F.3 Real World

To make sure that our method can be scaled for long-term tasks, we used our framework to control a real robot to carry out consecutive tasks. For the purposes of this experiment, we assign a goal

Environment	TACO-RL	LMP [13]	CQL [6]
maze2d-umaze	**110**±2.2	81.9±14.9	5.7
maze2d-medium	**88.9**±2	77±18	5.0
maze2d-large	**76.7**±9.7	20.1±29.3	12.5

Table 5. The average normalized score for three different random seeds. The CQL results were obtained from the D4RL whitepaper [54].

image for each of the robot's tasks. Our agent, which was trained entirely from unlabeled play data, can complete all of these sequential tasks by inferring how to transition between them, achieving the state depicted by the goal image. Examples of these sequential tasks are listed below:

- Moving the sliding door to the right and then opening the drawer
- Lifting the block and putting it on top of the drawer
- Lifting the block and placing it on a plastic container
- Turning the blue light on and then opening the drawer
- Turning the green light on and then open the drawer

G Negative Results

We present an incomplete list of experiments tried throughout the course of the research project. These ideas were tried a couple of times, but they in general do not improve performance. It is possible that they could work better with a more thorough investigation. We hope this experience will be helpful to researchers building on top of this work.

- Predicting absolute actions instead of relative actions with the decoder decreases the low-level policy performance. We believe that using relative actions might be easier for the agent to learn, as it does not have to memorize all the locations where an interaction has been performed.
- Decoder designed only with fully connected layers instead of a recurrent neural network lead to a slightly worse performance. We speculate that this could be due to the environment not completely following the Markov assumption.
- Using a gaussian mixture model instead of a mixture of logistics to predict the decoder actions lead to a similar performance.
- Using a pretrained ResNet18 in the imagenet dataset as visual encoder in simulation decreases performance for the low-level policy. More experiments by using the R3M pretrained model in simulation might offer different results.
- In the real world experiments, training the visual encoder from scratch for the static camera network decreases performance for the low-level policy. This can be explained due to the small amount of data present in the real-world dataset.

H Limitations

Despite the promising ability to learn diverse goal-reaching tasks even reasoning over long-horizon tasks, our method has a few aspects that warrant future research. Specifying a task to the goal-conditioned policy, requires providing a suitable goal image at test-time, which should be consistent with the current scene. An exciting direction for future work is to use natural language processing techniques to command the robot policy [55, 49]. If one wishes to sequence various tasks in the real world, an open question we did not address in this work is tracking task progress, in order to know when to move to the next task. In this work we acted with a fixed time-horizon for sequencing tasks in the real world, but this implicitly assumes that all tasks take approximately the same timesteps to complete. In addition, recent results [14] show that, in general, pure offline RL methods tend to offer better data-efficiency compared to behavioral cloning, which could be counted as one general limitation of imitation learning methods and derivatives of it, as the approach introduced in this work. However, since one of the most appealing properties of play lies in the simplicity of data collection, this limitation appears to be rather minor. Our method has a specific limitation in that we must first train the low-level policy before training the high-level policy, which requires more training time than training them together. However, because we train our approach offline, the robot does not need to perform rollouts in the environment during this time, making this a minor issue.