3 System Overview

Now that we have covered the necessary background knowledge for our system, we will focus on the system overview. First, we show how the policy interacts with the environment, followed by an overview of the policy network. Finally, we provide an overview of the training procedure and how policies are evaluated.

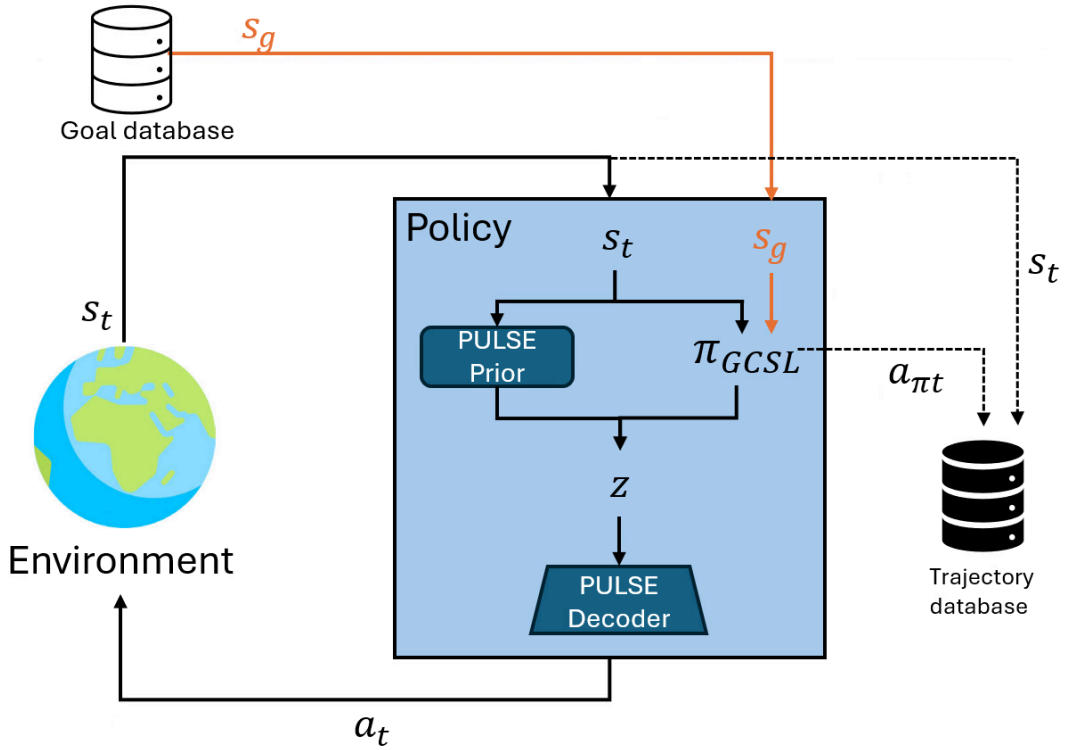

Figure 3: Overview of the hierarchical policy

Starting with a simulator. We use IsaacGym simulator for simulating the environment where the humanoid is in. The simulator is run at 60hz, and the policy is run at 30hz. The full-body humanoid pose in the simulation is represented by 3D joint rotations and poses. The humanoid model consists of 24 joints, with 23 of them actuated by a proportional derivative (PD) controller, where the action a_t specifies the PD targets. Movement is characterized by angular and linear velocities, while the physics simulation governs the state and transition dynamics. The policy determines the actions, with an action space dimension of $\mathbb{R}^{23x3}$, corresponding to the control inputs for the actuated joints.

In our experiment we compare two policy frameworks: hierarchical policy that uses pre-trained low-level policy and a policy that does not use any pre-trained motion latent space.

The hierarchical policy framework is composed of three components: a prior, a high-level policy π_{GCSL}, and a low-level policy. The prior and low-level policy, adopted from the PULSE framework [Luo et al., 2024], are pre-trained. The policy takes as input the current state s_t and the goal state s_g. The current state s_t includes

current time step, the joint positions, the joint velocities, the joint rotations, the joint rotational velocities, and the height of the root. The goal state, also known as keyframe, s_g is defined by joint positions and joint rotations. The goal state s_g is provided to the policy in egocentric coordinates, meaning the coordinates are first translated relative to the humanoid's root and then rotated accordingly. The high-level policy processes the goal state s_g alongside the current state s_t and generates a latent variable $a_{\pi t}$, which is combined with the prior's output as shown in equation 8. Then this latent variable is passed to the low-level policy, "PULSE decoder", which generates the corresponding action a_t.

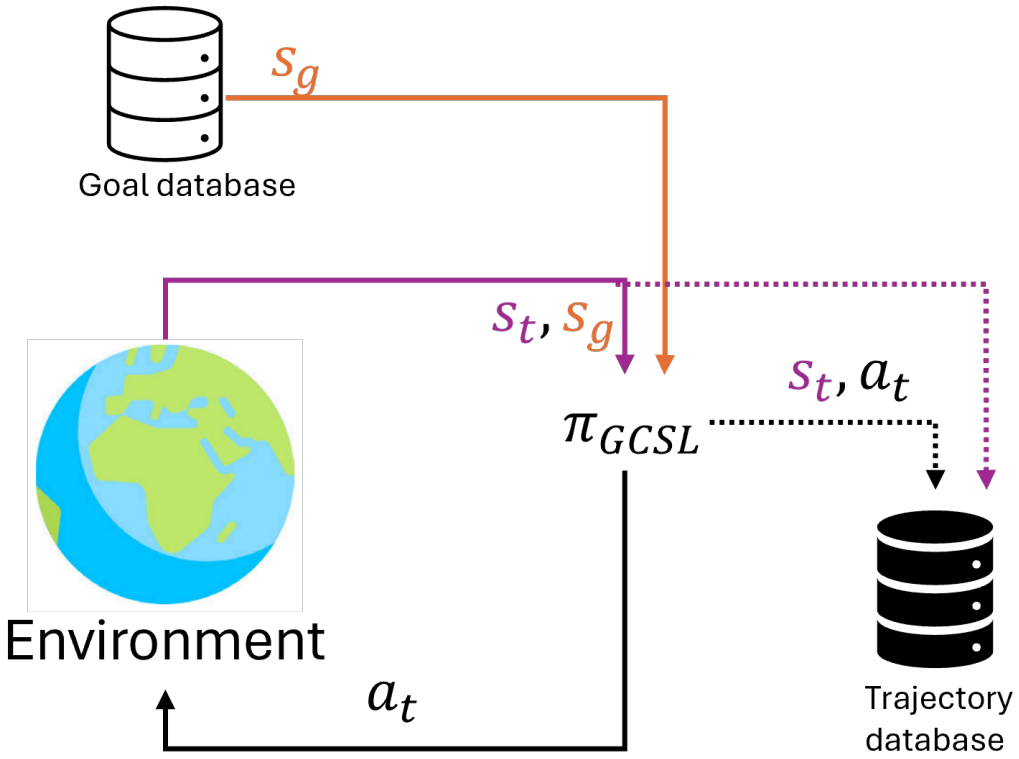

Figure 4: Overview of the plain policy

The plain policy consists of a single component: the policy itself. It shares the same input structure as the hierarchical policy, which includes the current state s_t and the goal state s_g. Based on this input, the policy outputs an action a_t. The key distinction between the plain policy (π_{GCSL}) and the hierarchical policy (π_{GCSL}) lies in the action dimensionality. Specifically, the plain policy operates in an action space of $\mathbb{R}^{69}$, whereas the high-level policy has a different action dimensionality, $\mathbb{R}^{32}$.

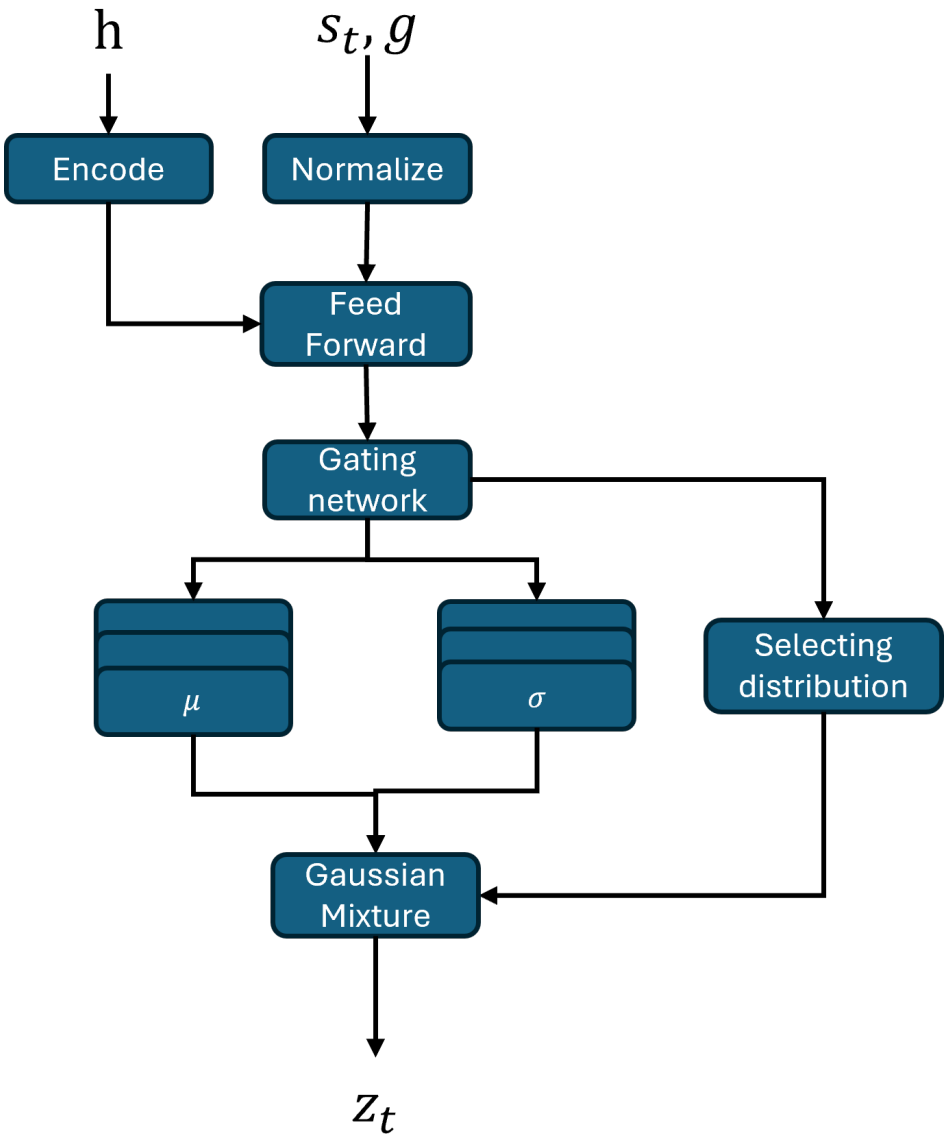

Figure 5: Overview of the policy network

We train a high-level policy and a plain policy. The both policies share the same neural network structure. The policy π_{GCSL} is a Gaussian mixture model that is represented as a neural network. The input for the network is the horizon h, the current state s_t which the agent is in, and the goal position g that the humanoid needs to achieve in the last time step. Since we want the agent to reach the goal in the last time step, the horizon is the time difference between the current time step and the episode length. The inputs s_t and g are first z-score normalized and then concatenated with the one-hot encoded horizon and fed into the feed forward layer.

The feed forward layer is represented as a three-layer MLP followed by the μ network that learns mean of the Gaussian and the σ network that learns variance of the Gaussian. μ and σ are structured as a mixture of experts framework [Jordan and Jacobs, 1993], which are five-layered MLPs. The amount of experts influence on the

final distribution is chosen by the gating network. The layers are activated with a ReLU activation function, except in the mixture of experts framework the layers are activated with a leaky-ReLU activation function. Finally, if the policy is gathering trajectories for the trajectory database, the action is chosen by sampling the Gaussian mixture model, but if the model is being validated, the action is chosen for each dimension from the most probable expert.

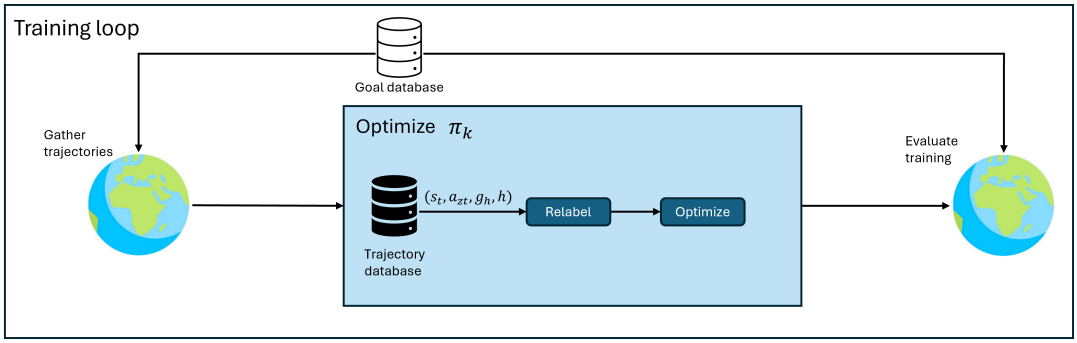

Figure 6: Training overview

We train both policies with GCSL. Trajectories are collected by commanding the agent to reach the goal state. The collected trajectories are added to the trajectory database and then relabeled as seen in Equation 3. When training the policy, relabeling is done on the fly. For each gradient step, we uniformly sample triplet that contains the index of the trajectory, index of the current state and index of the goal state on the trajectory. This reduces the memory load, since we only need to store the trajectories that are collected.

The objective for training the neural network is to minimize the negative log-likelihood of the Gaussian mixture over the relabeled data which can be understood as goal-conditioned behavior cloning. The objective is optimized with ADAM optimizer [Kingma and Ba, 2017]. To make the learning procedure more stable, cosine annealing is used to anneal the optimizers learning rate to zero during a single iteration of the training loop.

In our experiments, we only use one goal that is 1 meter away from the starting position. The policy is evaluated by commanding the policy to reach the given goal on the last time step. The full-body mean per joint position error (MPJPE) is used as the validation criterion. MPJPE is calculated as the mean of the sum of distances between corresponding joints:

$$E_{MPJPE} = \frac{1}{n_{joints}} \sum_{j \in joints} |g_j - s_j|_2 \tag{9}$$

Trajectories Collected per Iteration	Episode Length	Latent Dimensions	Gradient Steps per Iteration
3072	128	32	249984
Learning Rate	FF Layer Width	FF Layers	Experts
1×10^{-4}	512	3	4
MoE layer width	MoE layers	Gating Network Width	Gating Network Layers
512	5	512	3

Table 1: Hyperparameters for π_{GCSL}. Here FF is shorthand of feed forward and MoE is shorthand of mixture of experts

4 Evaluation

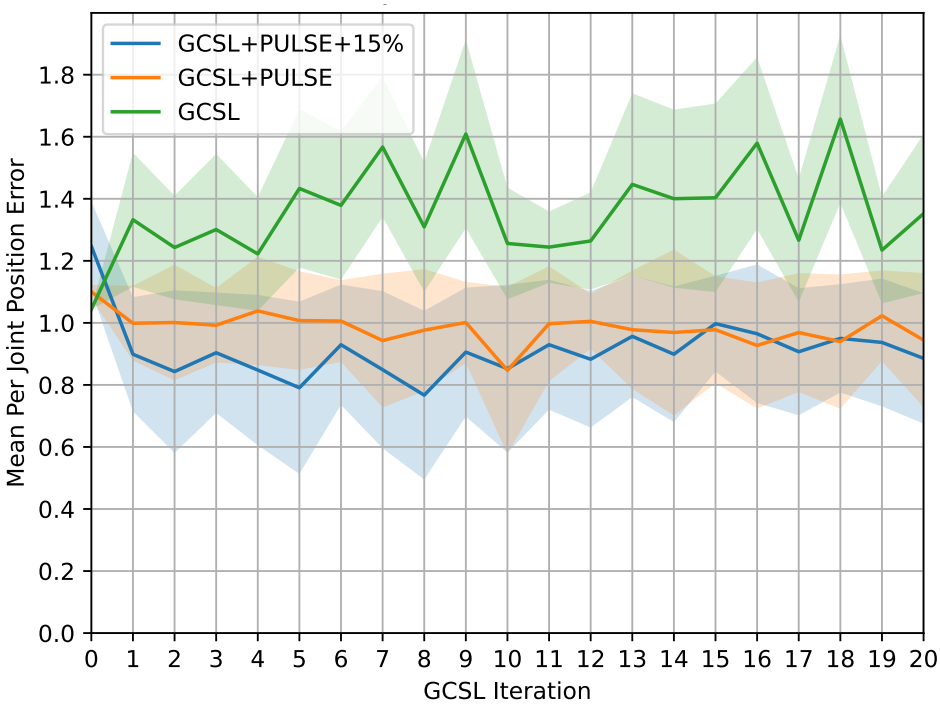

Figure 7: MPJPE across different policies. The solid line represents the mean of 128 evaluation trajectories, while the shaded region indicates the standard deviation. 'GCSL' corresponds to the policy trained from scratch, 'GCSL+PULSE' denotes the hierarchical policy, and 'GCSL+PULSE+15%' represents the hierarchical policy trained using only the top 15% of the best trajectories collected during each iteration.

This section investigates two aspects of policy training in reinforcement learning. First, we examine whether training a policy from scratch yields better performance compared to leveraging a pre-trained latent space. Second, we explore the impact of optimizing the training dataset by reducing it to include only the most successful trajectories. By focusing on the best performing trajectories, we aim to assess whether it is possible to achieve comparable policy performance with significantly fewer training samples, thereby improving model training efficiency.

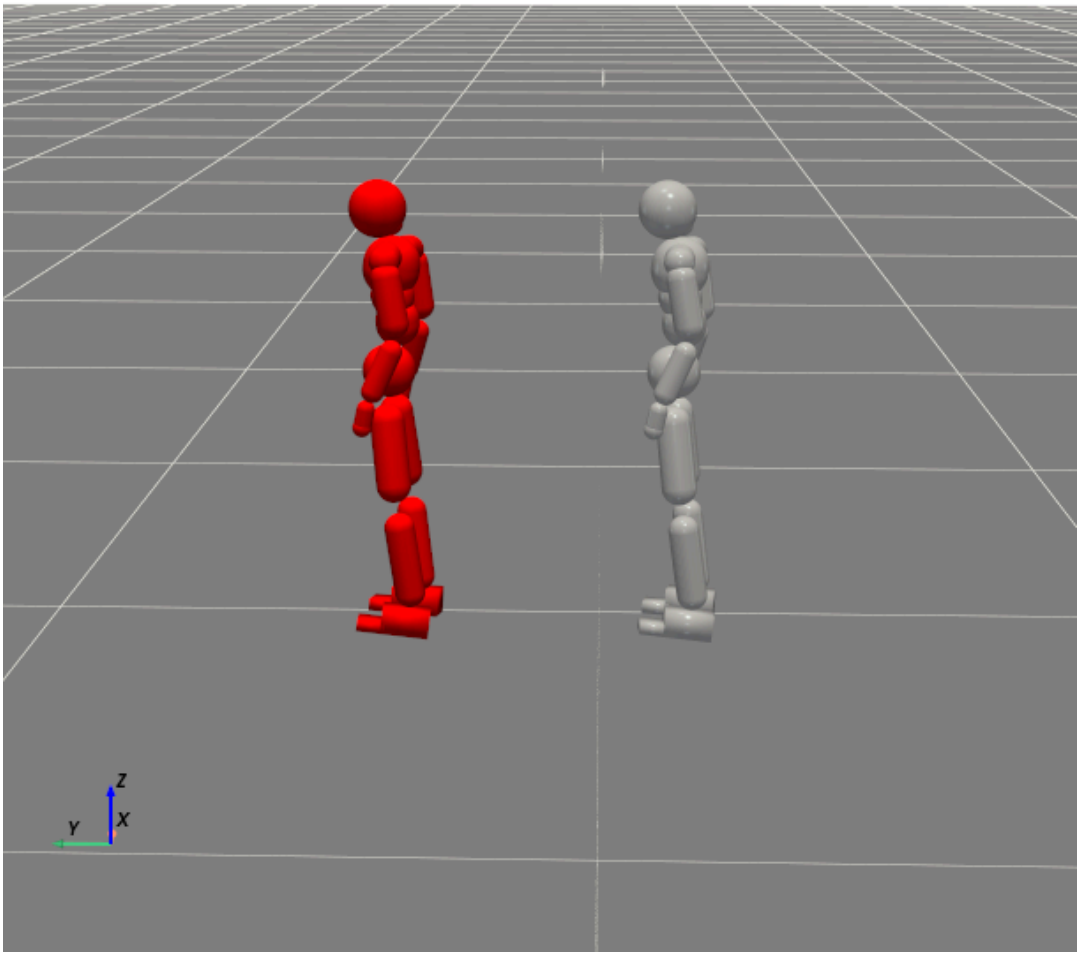

Figure 8: Experiment setup.

The Figure 9 shows the setup for our experiment. A white humanoid figure represents the starting position, while a red humanoid marks the goal state. The task is for the white-colored humanoid to move forward and reach position that is illustrated by the red-colored humanoid in the last time step of the episode. The distance between the initial position and goal position is 1m.

To achieve a policy that could reach to the goal position, we executed the training loop, illustrated in Figure 6, for 20 iterations for each policy. To compare the performance of different policies we plotted the evaluation results in Figure 7.

From Figure 7 can be observed that the hierarchical policy outperforms the policy trained from scratch. However, there is no significant performance difference between the two hierarchical policies. To further inspect the performance of the policies, figures 9, 10, and 11 were rendered. These renders show that the hierarchical policies were able to stay upward and reach closer to the goal pose than the policy trained from scratch. The main difference between hierarchical policies is that the policy

trained with the top 15% of the best trajectories collected during each iteration trains significantly faster, as it requires accessing a smaller, more curated dataset.

We observed from the figures 9, 10, and 11 that the hierarchical policy that was trained with the top 15% of the best performing trajectories did not stop upon reaching a state near the goal. Additionally, most of the trajectories generated by both hierarchical policies remained close to the initial state. This behavior contrasts with the policy trained from scratch, which exhibited trajectories that had only laying positions. Consequently, the primary factor contributing to the performance difference between the two approaches is that the hierarchical policy was able to maintain an upright standing position, which significantly influenced its overall stability.

GCSL	48h
GCSL+PULSE	48h
GCSL+PULSE+15%	29h

Table 2: Training time of the policies

Although we demonstrated that better results can be achieved using a pre-trained latent space, the outcomes remain far from optimal. In the following discussion, we reflect on potential reasons for these suboptimal results.

First, IsaacGym is known to produce stochastic actions, meaning that executing the same action sequence in different environments does not consistently result in identical state trajectories for the agent. This increases the difficulty of the behavior cloning task, as we cannot ensure that the actions will reliably replicate the desired state trajectory.

Second, the pre-trained latent space is also stochastic, meaning that the same latent variable does not always reconstruct to the same action. While this would be less problematic without strict time constraints, our experiment is time-sensitive. Without these constraints, we could train a Gaussian policy and sample actions near the optimal action to replicate motions closely. However, given the time constraint, it is crucial that the latent variable at the reached state reliably drives the agent to the goal state within the specified time horizon, rather than approximating the goal at an arbitrary future time step.

And finally, the advice Andrychowicz et al. [2017] presented in their work was overlooked. Andrychowicz et al. [2017] advised that even if there is one goal that the experimenter is interested in, it is faster to train the model with multiple goals.

Figure 9: Rendering of 30 randomly sampled final poses achieved by the policy trained from scratch. The red figure is the goal position and white opaque figures are the final reached poses.

Figure 10: Rendering of 30 randomly sampled final poses achieved by the hierarchical policy. The red figure is the goal position and white opaque figures are the final reached poses.

Figure 11: Rendering of 30 randomly sampled final poses achieved by the hierarchical policy that was trained with the top 15% of the best performing trajectories. The red figure is the goal position and white opaque figures are the final reached poses.

5 Future Work

In this section, we examine potential future work to improve our system. We first inspect periodic autoencoders as way to form a latent space and then we continue to challenge the idea of the offline data set.

One possible reason for the failure of the developed controller may lie in the structure of the PULSE decoder. Solutions based on periodic autoencoders such as DeepPhase [Starke et al., 2022], FLD [Li et al., 2024] and Bi level motion imitation [Zhao et al., 2024] shape the motion space into a phase manifold. This allows the encoder to capture essential features and temporal dependencies of natural motions. Enforcing latent dynamics in this way enhances proficiency and generalization capabilities. A comparison between our hierarchical components and a hierarchical structure consisting of a high-level controller and periodic autoencoder as a low-level controller, would provide further insight into this question.

Another approach to addressing the uncertainty of the physics simulator is online fine-tuning. In this method, the policy is initially trained on an offline dataset and then refined in an online setting. Zhao et al. [2022] and Ball et al. [2023] enabled this transition from offline learning to online learning using value-based methods. They achieved this by first populating the replay buffer used in online training with offline data and gradually replacing it with online data. Zhao et al. [2022] reported that online fine-tuning helps mitigate distribution shift, which can occur when deploying an offline-trained policy in an online setting. In our experiment, we trained the policy offline. Moving to a fully online learning approach could help determine whether the observed problems come from the offline data.