A Appendix

A.1 Environment Settings

We choose five out of the nine tasks introduced in CausalWorld since the other four tasks have limited support for configuring the initial and goal states. Specifically, we enumerate these five tasks here: (1) `Reaching` requires moving a robotic arm to a goal position and reach a goal block; (2) `Pushing` requires pushing one block towards a goal position with a specific orientation (restricted to goals on the floor level); (3) `Picking` requires picking one block at a goal height above the center of the arena (restricted to goals above the floor level); (4) `Pick And Place` is an arena is divided by a fixed long block and the goal is to pick one block from one side of the arena to a goal position with a variable orientation on the other side of the fixed block; (5) `Stacking` requires stacking two blocks above each other in a specific goal position and orientation.

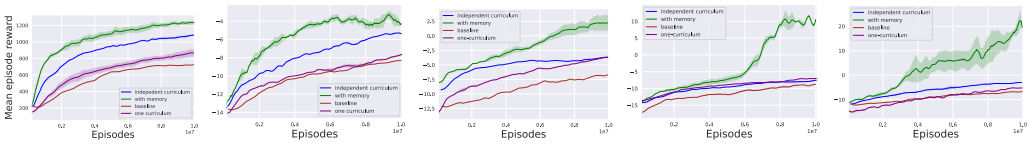
CausalWorld allows us to easily modify the initial states and goal states. In general, the initial state is the cylindrical position and Euler orientation of the block and goal state is the position variables of the goal block. These two control variables are both three-dimensional vectors with a fixed manipulation range. To match the range of each vector, we re-scale the generated initial states.

The reward function defined in CausalWorld is uniform across all possible goal shapes as the fractional volumetric overlap of the blocks with the goal shape, which ranges between 0 (no overlap) and 1 (complete overlap). We also re-scale the shaping reward to match this range.

We choose the PPO algorithm as our vanilla DRL policy learning method. We list the important hyper-parameters in Table. 1. We also provide the complete code in the supplementary material.

Table 1: Hyper-parameter values for PPO training

Parameter	Value
Discount factor (γ)	0.9995
n_steps	5000
Entropy coefficiency	0
Learning rate	0.00025
Maximum gradient norm	10
Value coefficiency	0.5
Experience buffer size	1e6
Minibatch size	128
clip parameter (ϵ)	0.3
Activation function	ReLU
Optimizer	Adam



(a) (b) (c) (d) (e)

Figure 4: Compare algorithms with different baselines in all of five tasks. Each learning curve is computed in three runs with different random seed.

A.2 Curricula Analysis and Visualization

In this section, we analyze the initial state curriculum and goal state curriculum. First, we replace the initial state curriculum with two different alternatives: (1) $MOC_{RandInitState}$, in which we replace the initial state curriculum in MOC with a uniformly chosen state. Other MOC components remains the same; (2) $MOC_{FixInitState}$, in which we replace the initial state curriculum in MOC with a fixed

	Mean Episode Reward	Success Ratio
$MOC_{RandInitState}$	936.9 ($\pm$35)	91% ($\pm$0.5%)
$MOC_{FixInitState}$	879.3 ($\pm$9)	89% ($\pm$1.1%)
$MOC_{RandGoalState}$	921.0 ($\pm$46)	91% ($\pm$0.5%)
MOC (Initial State)	1273 ($\pm$11)	100% ($\pm$0%)

(a) Analysis of initial state curriculum

	Mean Episode Reward	Success Ratio
GoalGAN	609 ($\pm$23)	56% ($\pm$18%)
ALP-GMM	568 ($\pm$26)	39% ($\pm$28%)
MOC (Goal State)	714 ($\pm$14)	68% ($\pm$15%)

(b) Analysis of subgoal curriculum

Table 2: Analysis of initial state curriculum and subgoal state curriculum.

initial state. The other MOC components remains the same. (3) $MOC_{RandGoalState}$, in which we replace the goal state curriculum in MOC with a uniformly chosen state. The other MOC components remains the same. The evaluations are conducted on the `reaching` task and the results are shown in Table 2a. From this table, we observe that MOC with initial state curriculum outperforms other two baseline schemes in terms of mean episode rewards and success ratio. This demonstrates the effectiveness of providing initial state curriculum. Besides, since "random sampling" outperforms "fixed initial state", we conjecture that it is better to provide different initial states, which might be beneficial for exploration.

In Sec. 5.1, we show that providing multi-objective curricula can improve the training of DRL agents. To further evaluate the advantages of hyper-RNN base-RNN framework, we conduct an experiment with GoalGAN, ALP-GMM and MOC with goal curriculum only. We evaluate on `reaching` task and the results are shown in Tab. 2b. In this table, we see that MOC Goal State (MOC$_{Memory^-,Goal^+}$), which is MOC has goal curriculum but doesn't have memory component, slightly outperform other two baseline schemes.

A.3 Additional Experimental Results

This section serves as a supplementary results for Sec. 5.

Fig. 5 shows the results of with and without Hyper-RNN in `pushing` tasks. The results validate the effectiveness of using Hyper-RNN. It is clear that, the incorporation of memory module consistently helps the DRL agent outperform other strong baselines in all scenarios. More importantly, in `pushing` task, we can observe a 5-fold improvement compared to the method with only the Hyper-RNN component.

Fig. 5 clearly validate the effectiveness of our proposed method in achieving both the best final performance and improving sample efficiency.

A.4 Additional Visualizations of States

Figs. 6, 7, 8, 9 visualize the state visitation density in task `reaching`, `picking`, `pushing` and `pick and place`, respectively.

From these results, we summarize the following observations: (1) The proposed architecture can help the agent explore different state spaces, which can be seen in the top row and bottom row. (2) The ablation study with three independent curricula often leads to exploring three different state space, as shown in Fig. 7 and Fig. 8. (3) By adding a memory component, the proposed MOC DRL can effectively utilize all curricula and help the agent focus on one specific state space. This is the reason why the proposed MOC DRL outperforms the other baselines in all tasks. (4) Comparing with Hyper-RNN ("no-mem") and without Hyper-RNN ("independent"), we can see that one of the benefits of using Hyper-RNN is aggregating different curricula. These can also be found in Fig. 7 and Fig. 8.

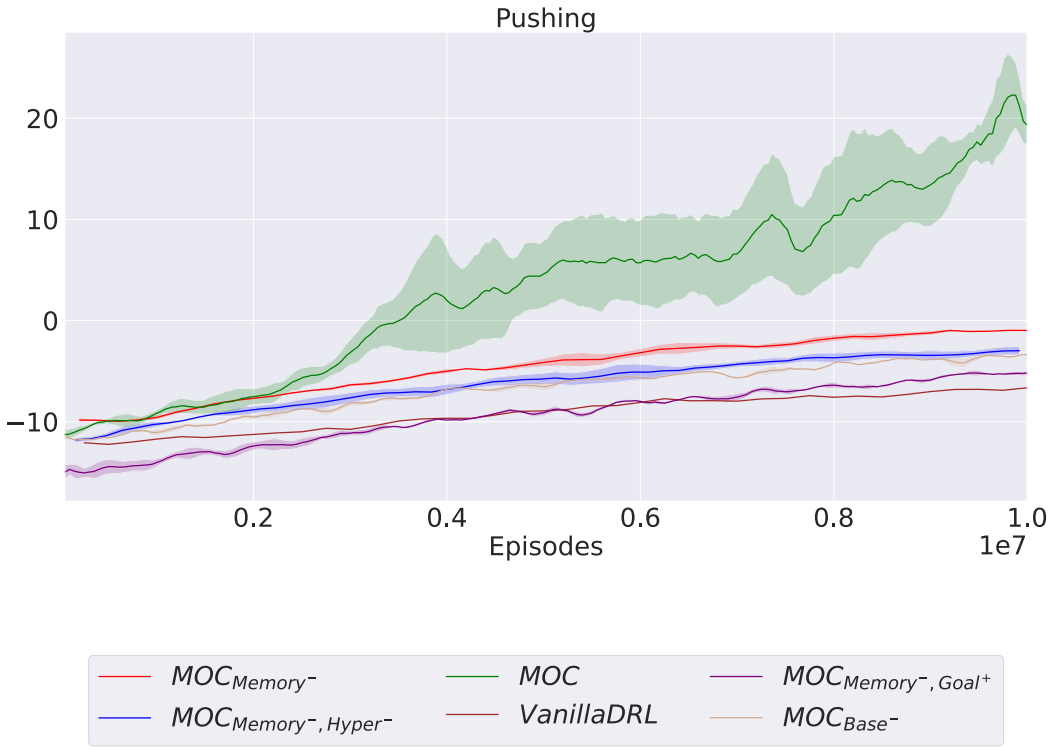
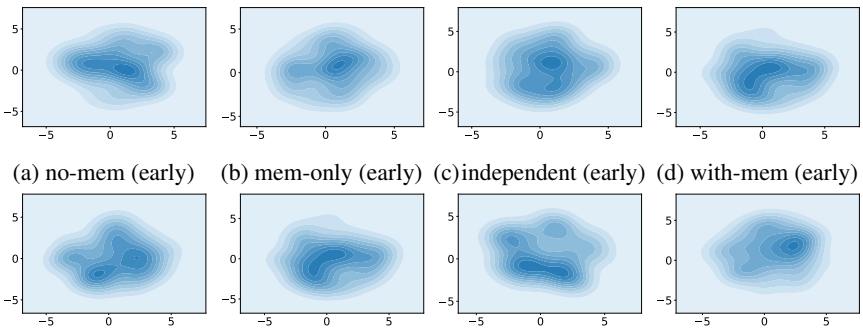


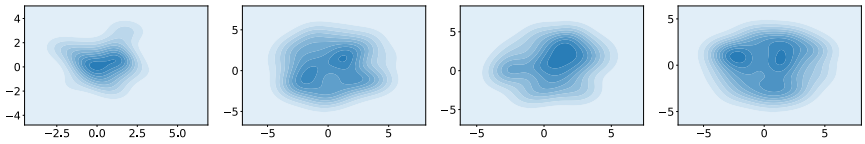
Figure 5: Comparison of algorithms with and without memory component in *pushing*. Each learning curve is computed in three runs with different random seeds.



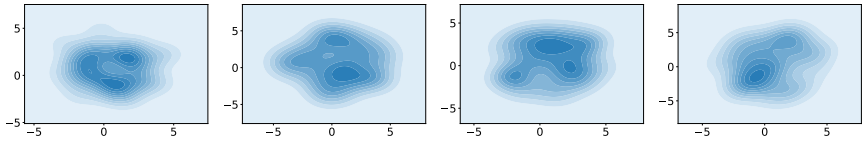
(a) no-mem (early) (b) mem-only (early) (c) independent (early) (d) with-mem (early)

(e) no-mem (late) (f) mem-only (late) (g) independent (late) (h) with-mem (late)

Figure 6: Visualizations of state visitation density in early and late stages in *reaching*

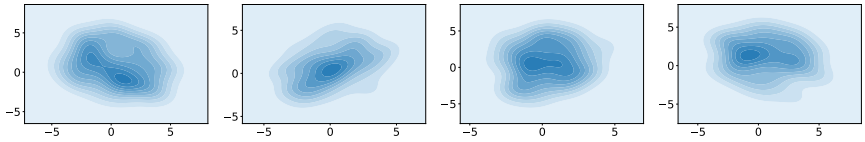


(a) no-mem (early) (b) mem-only (early) (c)independent (early) (d) with-mem (early)

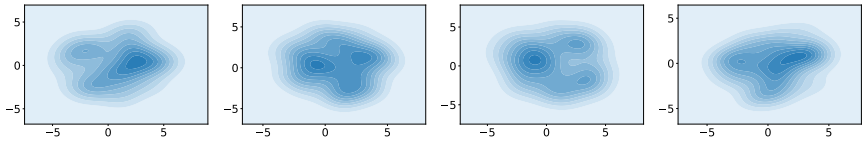


(e) no-mem (late) (f) mem-only (late) (g) independent (late) (h) with-mem (late)

Figure 7: Visualizations of state visitation density in early and late stages in *picking*



(a) no-mem (early) (b) mem-only (early) (c)independent (early) (d) with-mem (early)



(e) no-mem (late) (f) mem-only (late) (g) independent (late) (h) with-mem (late)

Figure 8: Visualizations of state visitation density in early and late stages in *pushing*

A.5 Additional Experiment Results

In Sec. 5.1, we compared MOC with state-of-the art ACL algorithms. Here, we add two more baselines algorithms. The results are shown in Fig. 13:

- InitailGAN [6]: which generates adapting initial states for the agent to start with.
- $PPO_{Reward^{+}}$: which is a DRL agent trained with PPO algorithm and reward shaping. The shaping function is instantiated as a deep neural network.

A.6 PPO Modifications

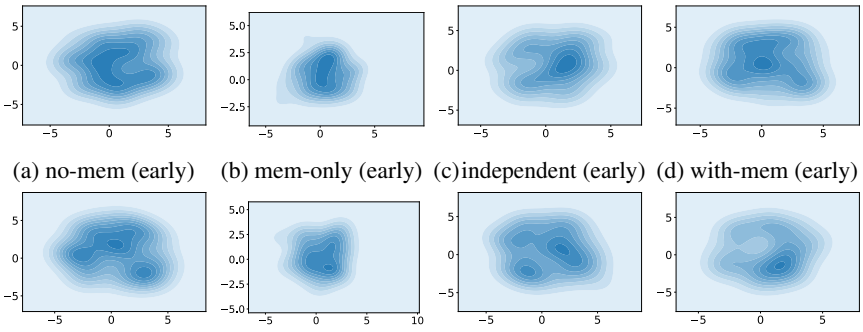
In Sec. 4, we propose a MOC-DRL framework for actor-critic algorithms. Since we adopt PPO in this paper, we now describe how we modify the PPO to cope with the learned curricula. We aim to maximize the PPO-clip objective:

$$\theta_{k+1} = argmax_{\theta}\mathbb{E}_{s,a\sim\pi_{\theta_k}}[min(\frac{\pi_{\theta}(a|s)}{\pi_{\theta_k}(a|s)}A^{\pi_{\theta_k}}(s,a), g(\epsilon, A^{\pi_{\theta_k}}(s,a))], \tag{2}$$

where

$$g(\epsilon, A) = \begin{cases} (1+\epsilon)A & A \geq 0 \\ (1-\epsilon)A & A < 0, \end{cases}$$

where θ is the parameter of policy π, θ_k is the updated k step parameter by taking the objective above, A is the advantage function that we define as:



(a) no-mem (early) (b) mem-only (early) (c)independent (early) (d) with-mem (early)

(e) no-mem (late) (f) mem-only (late) (g) independent (late) (h) with-mem (late)

Figure 9: Visualizations of state visitation density in early and late stages in *pick* and *place*

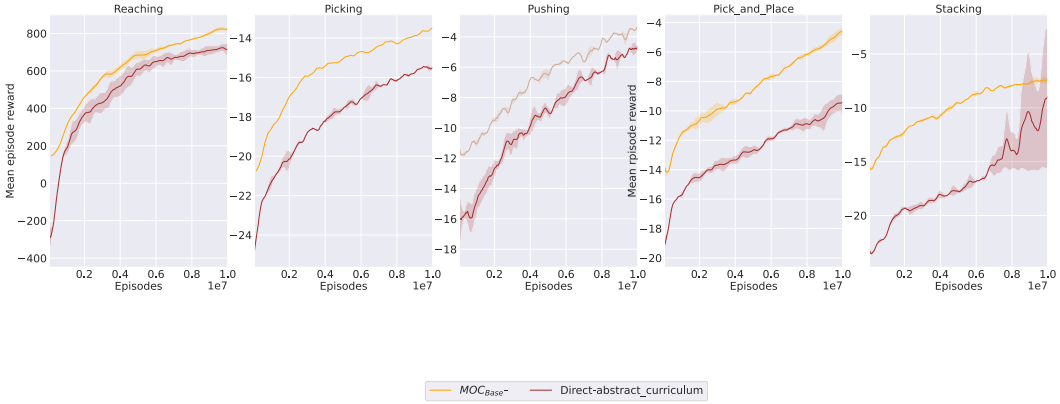


Figure 10: Comparison between read memory from memory and direct generate abstract curriculum

$$A(s,a) = Q(s,a) - V(s)$$

For the Hyper-RNN training, we modify the Q function as $Q(s, a, \mathbf{c}_{goal}, \mathbf{c}_{rew}, \mathbf{c}_{ini}, \mathbf{c}_{abs})$.

A.7 Bilevel Training

Here we provide more details regarding the bilevel training of Hyper-RNN introduced in Sec. 4.3. The optimal parameters θ_h^* are obtained by minimizing the loss function $\mathcal{J}_{outer}$. The key steps can be summarized as:

Step 1 Update PPO agent parameters θ on one sampled task by Eqn. 2

Step 2 With updated parameters θ, we train model parameters θ_h via SGD by minimizing the outer loss function $\theta_h^* = argmin_{\theta_h} \mathcal{J}_{outer}$.

Step 3 With θ_h, we generate manually designed curricula and abstract curriculum.

Step 4 We give the generate curriculum to the Q function and environment hyper-parameters.

Step 5 We go back to **Step 1** for agent training until converge.

A.8 Hyper-net

[14] introduce to generate parameters of Recurrent Networks using another neural networks. This approach is to put a small RNN cell (called the Hyper-RNN cell) inside a large RNN cell (the main RNN). The Hyper-RNN cell will have its own hidden units and its own input sequence. The input sequence for the Hyper-RNN cell will be constructed from 2 sources: the previous hidden

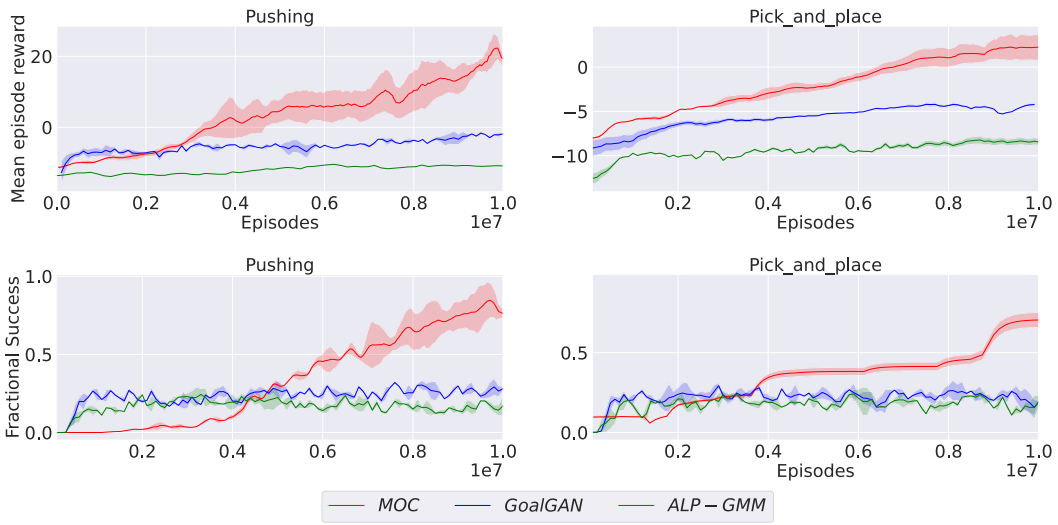


Figure 11: Comparison with ACL algorithms. Each learning curve is computed in three runs with different random seeds.

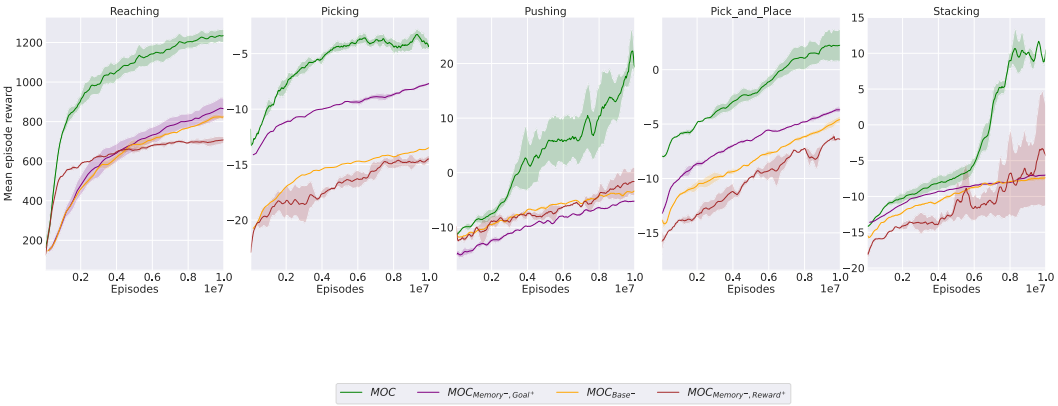


Figure 12: Comparison with reward curriculum only.

states of the main LSTM concatenated with the actual input sequence of the main LSTM. The outputs of the Hyper-RNN cell will be the embedding vector Z that will then be used to generate the weight matrix for the main LSTM. Unlike generating weights for convolutional neural networks, the weight-generating embedding vectors are not kept constant, but will be dynamically generated by the HyperLSTM cell. This allows the model to generate a new set of weights at each time step and for each input example. The standard formulation of a basic RNN is defined as:

$$h_t = \phi(W_h h_{t-1} + W_x x_t + b),$$

where h_t is the hidden state, ϕ is a non-linear operation such as $tanh$ or $relu$, and the weight matrics and bias $W_h \in \mathbb{R}^{N_h \times N_h}, W_x \in \mathbb{R}^{N_h \times N_x}, b \in \mathbb{R}^{N_h}$ is fixed each timestep for an input sequence $X = (x_1, \ldots, x_T)$. More conceretly, the parameters W_h, W_x, b of the main RNN are different at different time steps, so that h_t can now be computed as:

$$\begin{aligned} h_t &= \phi(W_h(z_h)h_{t-1} + W_x(z_x) + b(z_b)), where \\ W_h(z_h) &=< W_{hz}, z_h > \\ W_x(z_x) &=< W_{xx}, z_x > \\ b(z_b) &= W_{bz} z_b + b_0 \end{aligned} \tag{3}$$

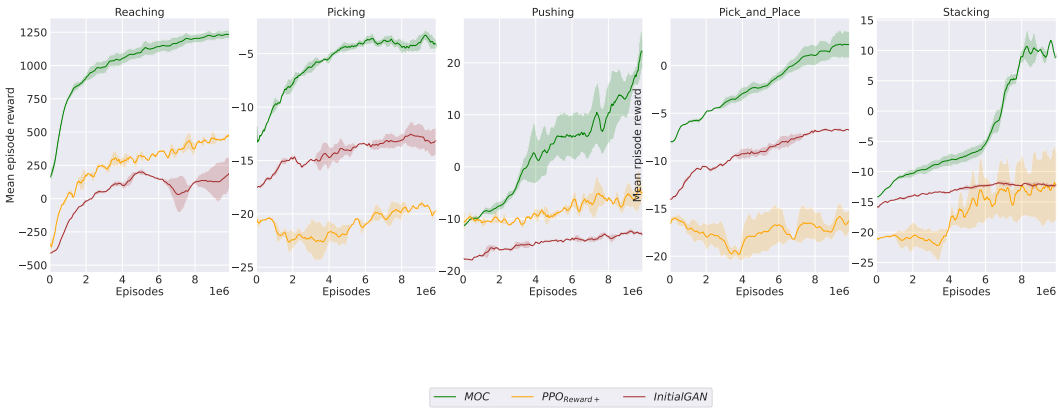


Figure 13: Comparison with Initial GAN and PPO with reward shaping only.

where $W_{hz} \in \mathbb{R}^{N_h \times N_h \times N_z}$, $W_{xz} \in \mathbb{R}^{N_h \times N_x \times N_z}$, $W_{bz} \in \mathbb{R}^{N_h \times N_z}$, $b_o \in \mathbb{R}^{N_h}$ and $z_h, z_x, z_z \in \mathbb{R}^{N_z}$. Moreover, z_h, z_x and z_b can be computed as a function of x_t and h_{t-1}:

$$
\begin{aligned}
\hat{x}_t &= \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \\
\hat{h}_t &= \phi(W_{\hat{h}} \hat{h}_{t-1} + W_{\hat{x}} \hat{x}_t + \hat{b}) \\
z_h &= W_{\hat{h}h} \hat{h}_{t-1} + \hat{b}_{\hat{h}h} \\
z_x &= W_{\hat{h}x} \hat{h}_{t-1} + \hat{b}_{\hat{h}x} \\
z_b &= W_{\hat{h}b} \hat{h}_{t-1}
\end{aligned}
\tag{4}
$$

Where $W_{\hat{h}} \in \mathbb{R}^{N_{\hat{h}} \times N_{\hat{h}}}, W_{\hat{x}} \in \mathbb{R}^{N_{\hat{h}} \times (N_h + N_z)}, b \in \mathbb{R}^{N_{\hat{h}}}$, and $W_{\hat{h}h}, W_{\hat{h}x}, W_{\hat{h}b} \in \mathbb{R}^{N_z \times N_{\hat{h}}}$ and $\hat{b}_{\hat{h}h}, \hat{b}_{\hat{h}x} \in \mathbb{R}^{N_z}$. The Hyper-RNN cell has $N_{\hat{h}}$ hidden units.

A.9 The abstract curriculum training

For some difficult tasks, we find that it is difficult to train a policy with small variances if the Hyper-RNN is initialized with random parameters[3].

As a simple workaround, we propose to pre-train the Hyper-RNN and memory components in a slightly unrelated task. In particular, when solving task T_x, we pre-train the abstract memory module on tasks other than T_x.

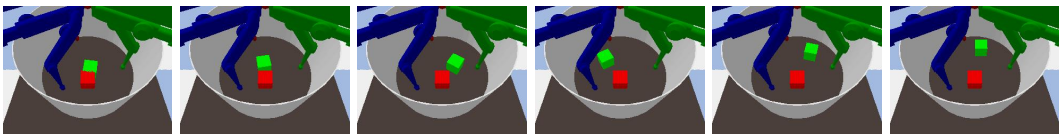
A.10 The visualization of generated sub-goal

The visualization of generated sub-goal state is shown in Fig. 14. Specifically, the arm is tasked to manipulate the red cube to the position shown as a green cube. As we can see, MOC generates subgoals that gradually change from "easy" (which are close to the initial state) to "hard" (which are close to the goal state). The generated subgoals have different configurations (*e.g.*, the green cube is headed north-west in 7000k steps but is headed north-east in 9000k steps), which requires the agent to learn to delicately manipulate robot arm.

A.11 Hyperparameters

In this section, we extensively evaluate the influence of different hyperparameters for the baselines and MOC, where the search is done with random search. We choose the reaching and stacking tasks, which are shown in Fig. 15, 16, 17. For example, in Fig. 15-(a), the first column represents the

[3]The weight initialization approach [65] designed for hyper-net does not help too much in our case.

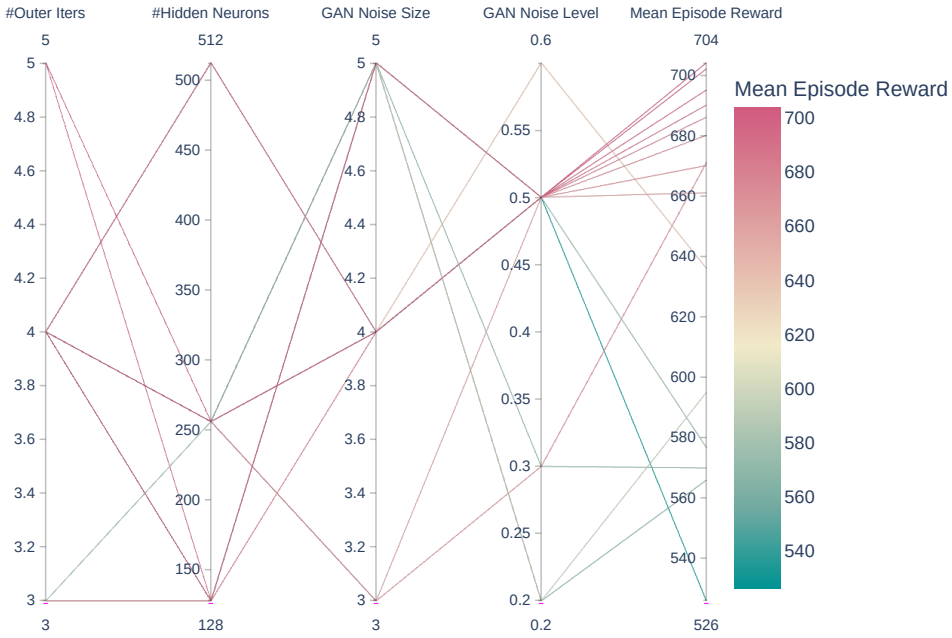


(a) 1000k steps (b) 3000k steps (c) 5000k steps (d) 7000k steps (e) 9000k steps (f) Goal state

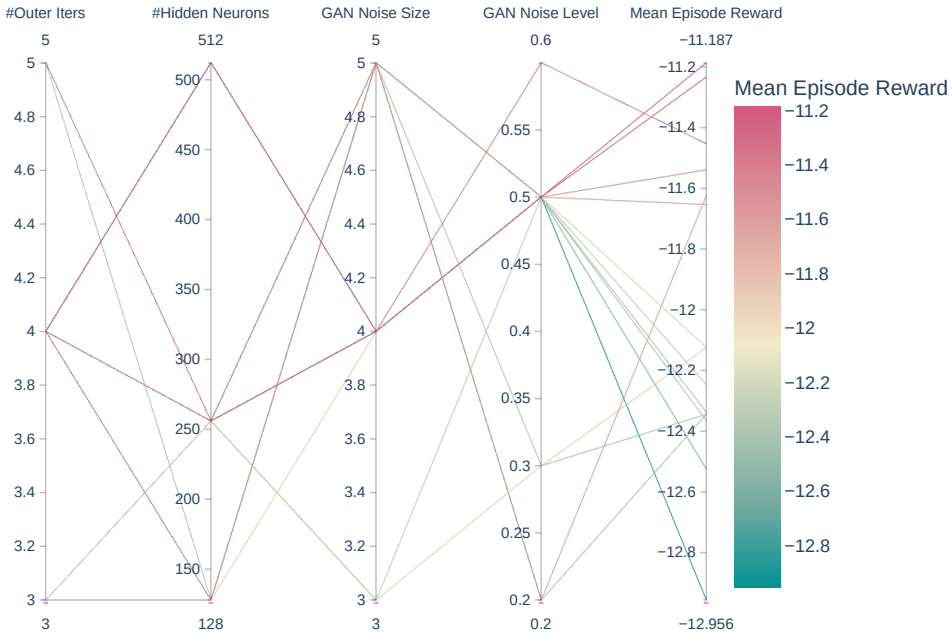
Figure 14: Visualization of generated subgoals

different values for outer iterations. A particular horizontal line, *e.g.*, $\{4, 512, 5, 0.5\}$, indicates a particular set of hyperparameters for one experiment. Besides, during the training phase, we adopt hyperparameters of PPO from stable-baselines3 and search two hyperparameters to test the MOC sensitivity.

We can observe that: (1) It is clear that MOC outperforms all the baselines with extensive hyperparameter search. (2) MOC is not sensitive to different hyperparameters.

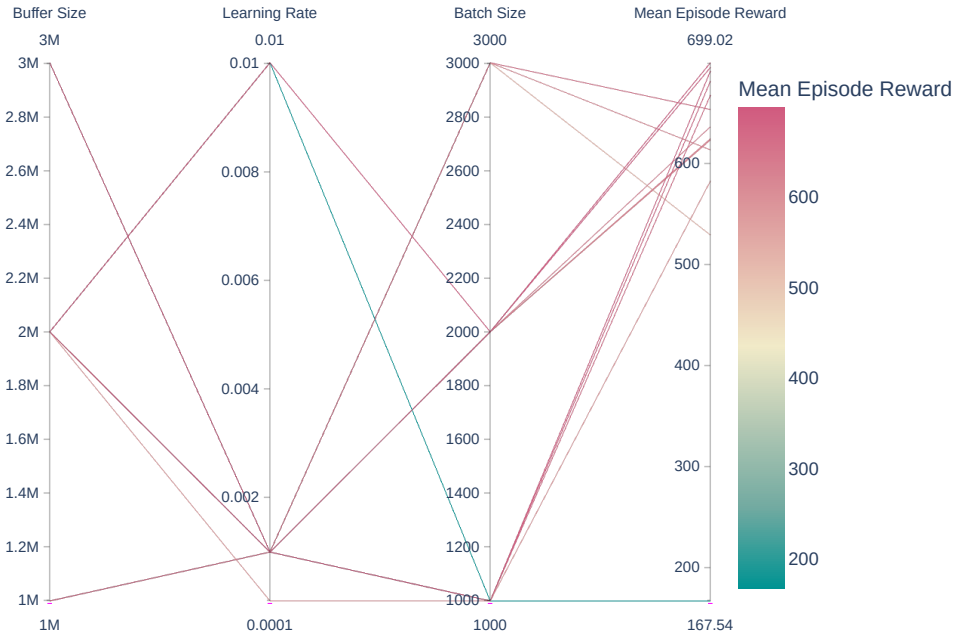


(a) Hyperparameter tuning in reaching task.

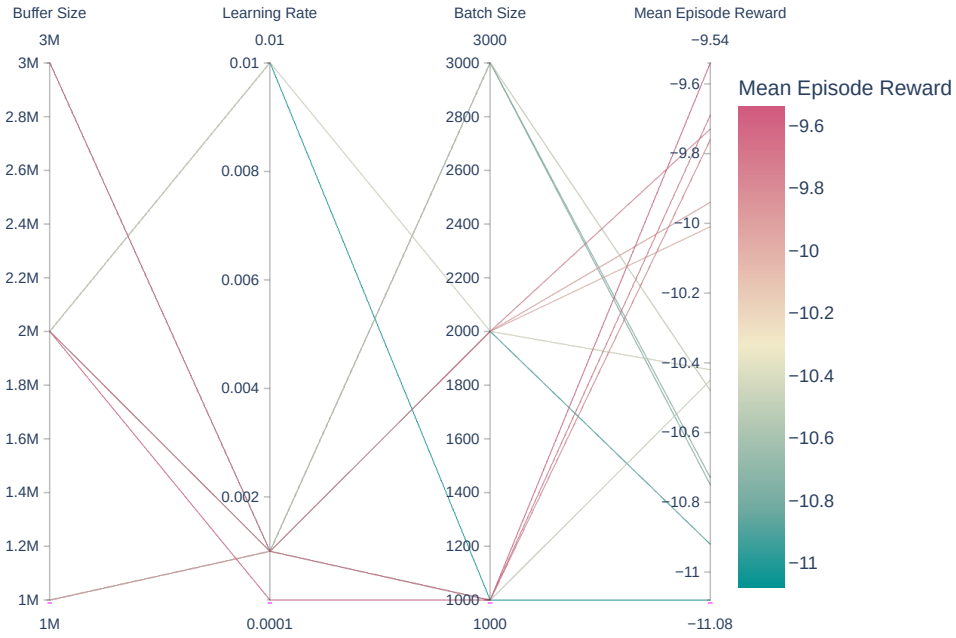


(b) Hyperparameter tuning in stacking task.

Figure 15: Hyperparameter tuning results for GoalGAN

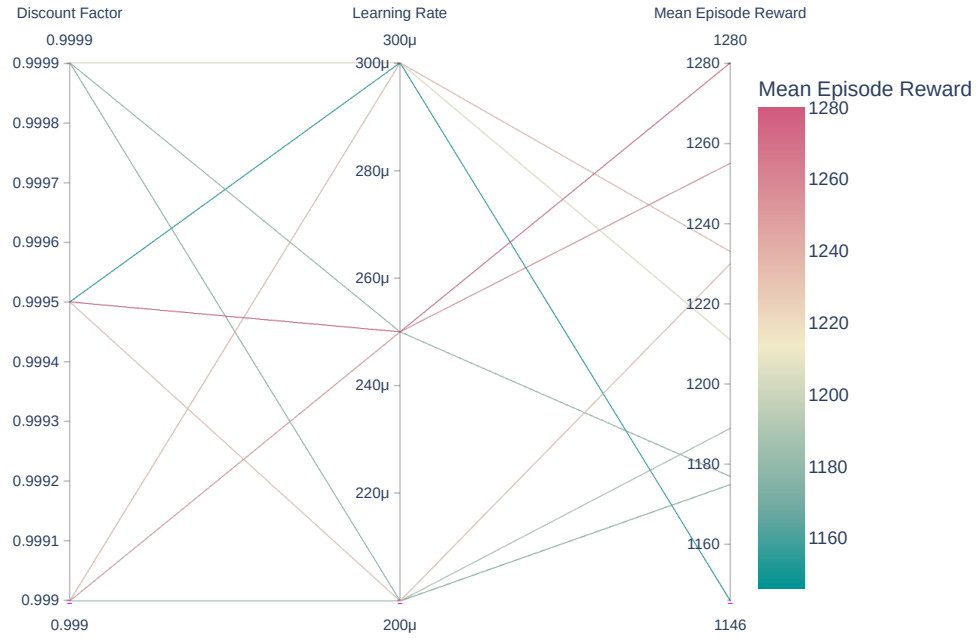


(a) Hyperparameter tuning in reaching task.

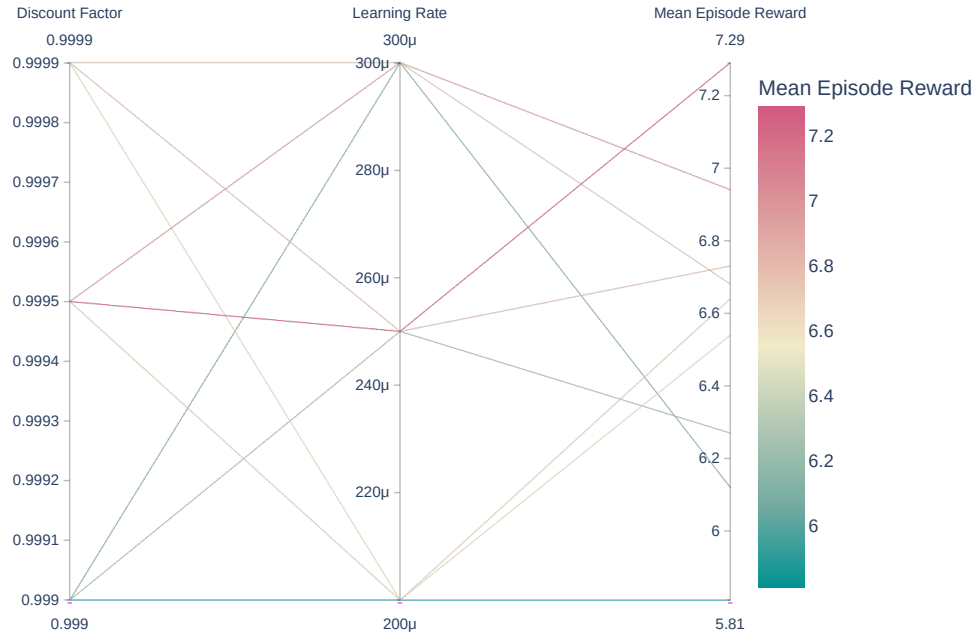


(b) Hyperparameter tuning in stacking task.

Figure 16: Hyperparameter tuning results for ALP-GMM



(a) Hyperparameter tuning in reaching task.



(b) Hyperparameter tuning in stacking task.

Figure 17: Hyperparameter tuning results for MOC