Learning World Models for Unconstrained Goal Navigation

Yuanlin Duan
Rutgers University
yuanlin.duan@rutgers.edu

Wensen Mao
Rutgers University
wm300@cs.rutgers.edu

He Zhu
Rutgers University
hz375@cs.rutgers.edu

Abstract

Learning world models offers a promising avenue for goal-conditioned reinforcement learning with sparse rewards. By allowing agents to plan actions or exploratory goals without direct interaction with the environment, world models enhance exploration efficiency. The quality of a world model hinges on the richness of data stored in the agent's replay buffer, with expectations of reasonable generalization across the state space surrounding recorded trajectories. However, challenges arise in generalizing learned world models to state transitions backward along recorded trajectories or between states across different trajectories, hindering their ability to accurately model real-world dynamics. To address these challenges, we introduce a novel goal-directed exploration algorithm, MUN (short for "World Models for Unconstrained Goal Navigation"). This algorithm is capable of modeling state transitions between arbitrary subgoal states in the replay buffer, thereby facilitating the learning of policies to navigate between any "key" states. Experimental results demonstrate that MUN strengthens the reliability of world models and significantly improves the policy's capacity to generalize across new goal settings.

1 Introduction

Goal-conditioned reinforcement learning (GCRL) has emerged as a powerful framework for learning diverse skills within an environment and subsequently solving tasks based on user-specified goal commands, without requiring further training (Mendonca et al., 2021; Andrychowicz et al., 2017). Given that specifying dense task rewards for GCRL requires domain expertise, access to object positions, is time-consuming, and is prone to human errors, rewards in GCRL are typically sparse, signaling success only upon reaching goal states. However, sparse rewards pose a challenge for exploration during training. To address this challenge, several previous methods, e.g., (Hafner et al., 2019a; Hansen et al., 2023; Mendonca et al., 2021) have proposed learning a generative world model of the environment using a reconstruction (decoder) objective, an instantiation of Model-based Reinforcement Learning (MBRL), visualized in Fig. 1. This approach is appealing because the world model can provide a rich learning signal (Yu et al., 2020; Georgiev et al., 2024). For example, world models allow agents to plan their actions or exploratory goals without directly interacting with the real environment for more efficient exploration (Hu et al., 2023; Sekar et al., 2020).

Existing MBRL techniques train world models to capture the dynamics of the environment from the agent's past experiences stored in a replay buffer. The richness of the data stored in the agent's replay buffer directly impacts the quality of a World Model. It is expected that the world model generalizes reasonably well to the state space surrounding the trajectories recorded in the replay buffer. However, the world model may not generalize well to state transitions backward along recorded trajectories or to states across different trajectories, which impedes the world model's learning of the real-world dynamics.

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

To induce a data-rich replay buffer covering a wide range of dynamic transitions, in this paper, we present a novel goal-directed exploration algorithm for effective world modeling and policy learning, MUN (short for "World **M**odels for **U**nconstrained Goal **N**avigation"). MUN facilitates modeling state transitions between any subgoal states in the replay buffer, whether tracing back along recorded trajectories or transitioning between

Figure 1: The general framework of model-based RL.

states on separate trajectories. This enhances the reliability of the learned world model and significantly improves the generalizability of the policy derived from the model to real-world environments, thereby boosting the exploration capabilities of the method. Additionally, we introduce a simple and practical strategy for discovering *key* subgoal states from the replay buffer. The key subgoals precisely mark the milestones necessary for task completion, such as steps like grasping and releasing blocks in the context of block-stacking scenarios. By world modeling and policy learning for unconstrained navigation between these key states, MUN can generalize to new goal settings, such as block unstacking that was not given to the agent at training time.

Our key contributions are as follows. First, we propose a novel goal-directed exploration algorithm MUN for effective world modeling of state transition between arbitrary subgoal states in replay buffers. As the quality of the world model improves, MUN becomes highly effective at learning goal-conditioned policies that excel at exploration in sparse-reward environments. Second, we present a practical strategy for identifying pivotal subgoal states, which serve as milestones in completing sophisticated tasks. By training world models for unconstrained transition between these milestones, our method enables learning policies that can adapt to novel goal scenarios. Finally, we evaluate MUN in challenging robotics environments, such as guiding a multi-legged ant robot through a maze, maneuvering a robot arm amidst cluttered tabletop objects, and rotating items in the grasp of an anthropomorphic robotic hand. Across these environments, MUN exhibits superior efficiency in training generalizable goal-conditioned policies compared to baseline methods and ablations.

2 Problem Setup and Background

We consider the problem of goal-conditioned reinforcement learning (GCRL) under a Markov Decision Process (MDP) parameterized by $(S, A, P, G, \eta, R, \rho_0)$. S and A are the state and action spaces, respectively. The probability distribution of the initial states is given by $\rho_0(s)$, and $P(s'|s, a)$ is the transition probability. $\eta : S \rightarrow G$ is a mapping from the state space to the goal space, which assumes that every state s can be mapped to a corresponding achieved goal g. The reward function R is defined as $R(s, a, s', g) = 1\{\eta(s') = g\}$. We assume that each episode has a fixed horizon T. For ease of presentation, we further assume $S = G$ and η is an identify function in this paper.

A goal-conditioned policy is a probability distribution $\pi : S \times G \times A \rightarrow \mathbb{R}^+$, which gives rise to trajectory samples of the form $\tau = \{s_0, a_0, g, s_1, \ldots, s_T\}$. The purpose of the policy π is to learn how to reach the goals drawn from the goal distribution p_g. With a discount factor $\gamma \in (0, 1)$, it maximizes $J(\pi) = \mathbb{E}_{g \sim p_g, \tau \sim \pi(g)} \left[\sum_{t=0}^{T-1} \gamma^t \cdot R(s_t, a_t, s_{t+1}, g) \right]$.

In the context of model-based reinforcement learning (MBRL), a world model $\hat{M}$ is trained over trajectories sampled from the agent's interactions with the real environment, which are stored in a replay buffer, to predict the dynamics of the real environment. Fig. 1 illustrates the general MBRL framework. We use the world model structure $\hat{M}$ of Dreamer (Hafner et al., 2019a,b, 2020, 2023) to learn real environment dynamics as a recurrent state-space model (RSSM). We provide a detailed explanation of the network architecture and working principles of the RSSM in Appendix A.1. Our study focuses on tackling the world model learning problem in goal-conditioned model based reinforcement learning settings. Particularly, we consider **GC-Dreamer** (goal-conditioned Dreamer) as an important baseline with the following learning components:

$$\text{World Model:} \quad \hat{M}(s_t|s_{t-1}, a_{t-1}) \qquad \text{Actor:} \quad \pi^G(a_t|s_t, g) \qquad \text{Critic:} \quad V(s_t, g) \qquad (1)$$

2

(a) Key subgoal states in a 3-Block Stacking task. (b) Bidirectional Replay Buffer
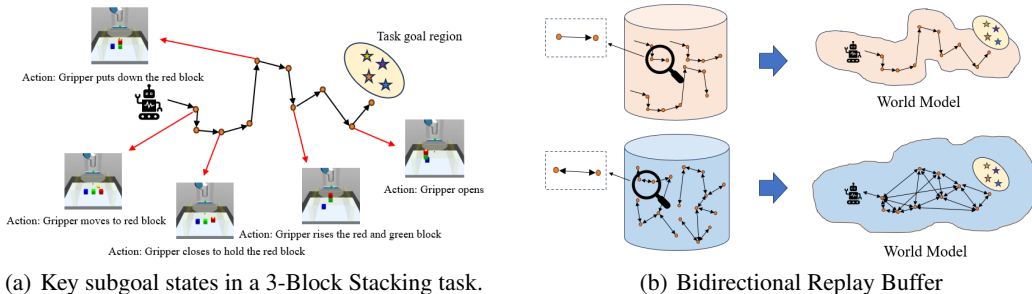
Figure 2: In Fig. 2(a), we illustrate the key states involved in completing the task of 3-block stacking. In Fig. 2(b), we demonstrate the significant advantages of the bidirectional replay buffer used in MUN over traditional methods in learning world models.

In GC-Dreamer, the goal-conditioned agent $\pi^G(a|s,g)$ samples goal commands $g \in G$ from the given environment goal distribution p_g to collect trajectories in the real world. These trajectories are used to train the world model $\hat{M}$, and subsequently, π^G is trained on imagined rollouts generated by $\hat{M}$ using the model-based actor-critic algorithm in Dreamer (Hafner et al., 2020), with these two steps run in alternation. The critic estimates the sum of future rewards $\sum_t r_t^G$, and the actor tries to maximize the predicted values from the critic. The goal-reaching reward r^G is defined by the self-supervised temporal distance network D_t (Mendonca et al., 2021), i.e. $r^G(s,g) = -D_t(s,g)$. D_t predicts the anticipated number of action steps needed to transition from s to g. Essentially, π^G is reinforced to minimize the action steps required to transition from the current state s to a sampled goal state g. The temporal distance estimator D_t is trained by extracting pairs of states s_t and s_{t+k} from an imagined rollout generated by running the policy over the world model and predicting the distance k between them as follows:

$$D_t\big(\Psi(s_t), \Psi(s_{t+k})\big) \approx k/H \tag{2}$$

Here, Ψ represents the preprocessing for imagined states, such as transforming them into the world model's latent space (we assume $S = G$ in the paper). H represents the total length of the imagined rollout. Further details on the training procedure of D_t can be found in Appendix A.2.

3 Training World Models for Unconstrained Goal Navigation

In this section, we introduce MUN, our main approach to addressing the core challenge in GCRL: efficient exploration in long-horizon, sparse-reward environments. Our approach focuses on enhancing the agent's understanding of the real-world environment through improved dynamic (world) modeling and latent space representation. As the quality of the world model improves, the goal-conditioned policy developed from it generalizes more effectively to the real environment. By closing the generalization gap between the policy's behavior in the real environment and the world model, MUN effectively guides the agent's exploration towards the desired goal region in the real environment.

3.1 Training Generalizable World Models

Fig 1 illustrates the general framework of Model-based RL, where world models are trained using agent's experiences stored in a replay buffer populated with observed environment transitions (s_t, a_t, s_{t+1}) linking the environment's future states s_{t+1} and past states s_t along with the corresponding control actions a_t. The richness of the environment space and dynamic transitions captured by the replay buffer define the extent of what a world model can learn about the real environment. Through supervised learning, the model can generalize reasonably well within the state space moving *forward* along the trajectories recorded in the replay buffer. However, it may be inaccurate for the state transitions moving *backward* along the recorded trajectories or *across* different trajectories. Consider the task of stacking blocks using a robot manipulator in Fig. 2(a). When humans learn to stack blocks, they also understand how to reverse the process to unstack the blocks or return to the initial state. In contrast, a world model trained solely on data from policies under training for stacking

3

Algorithm 1 The main training framework of MUN

1: **Input:** Policy π^G, World Model $\hat{M}$, reward function r^G, subgoals transfer number N_s, subgoal time limit T_s
2: Initialize buffers D, D_{DAD}, D_{egc}
3: **for** $i = 1$ to N_{train} **do**
4:    **if** Should Plan Subgoals **then**
5:        $B_{egc} \leftarrow$ A batch of episodes from D_{egc}
6:        $G_{subgoals} \leftarrow$ DAD(B_{egc}) with Algorithm 2
7:    Initialize empty trajectory τ
8:    **for** $s = 1$ to N_s **do**
9:        $t_s = 0$
10:        $g_s =$ Sample a subgoal randomly from $G_{subgoals}$
11:        **while** agent has not reached g_s and $t_s < T_s$ **do**
12:            Append one step in real environment with π^G using goal g_s to τ
13:            $t_s \leftarrow t_s + 1$
14:    $D_{DAD} \leftarrow D_{DAD} \cup \{\tau\}$
15:    $\tau' \leftarrow$ Trajectory of π^G sampled using the environment goal distribution $g \sim p_g$
16:    $D_{egc} \leftarrow D_{egc} \cup \tau'$
17:    $D \leftarrow D_{DAD} \cup D_{egc}$
18:    Update $\hat{M}$ with D
19:    Update π^G in imagination with $\hat{M}$ to maximize r^G

---

is unlikely to accurately model the unstacking process. As a result, the model may yield hallucinated trajectories for training policies, causing a significant discrepancy between the policy's behavior in the model and in the real world, thereby leading to ineffective exploration.

To improve model generalizability, in MUN, we proposed to learn world models capable of characterizing state transitions between any states in the replay buffer, whether by tracing back along recorded trajectories or transitioning between states on separate trajectories. Fig. 2(b) visualizes the comparison between the bidirectional replay buffer for learning world models used in MUN and the unidirectional replay buffer in conventional model-based algorithms. The bidirectional replay buffer not only covers a wider observation space but also captures a richer set of dynamic transitions. As discussed in Sec. 2, due to joint optimization, the richer set of dynamic transitions in MUN allows for a more reliable latent representation of the environmental space and consequently a higher quality reward function (Equation 2) for training policies generalizable to the real environment on top of the learned model.

We depict the learning algorithm in MUN in Algorithm 1. In the algorithm, we maintain $G_{subgoals}$ as a set of pivot subgoal states sampled from the relay buffer (illustrated in Algorithm 2) and aim to learn world models capable of seamless transitions between these subgoals. At line 6, we periodically update $G_{subgoals}$ as the training evolves. In the loop starting from line 8, we repeatedly sample N_s subgoals from $G_{subgoals}$ and direct the agent to sequentially reach these subgoals within a time limit of T_s steps for each. In this way, MUN samples a replay buffer that records bidirectional state transitions between the subgoals in $G_{subgoals}$. Based on our experience, we find that setting $N_s = 2$ is sufficient. Further discussion on the setting of N_s is provided in Sec 4. At line 18, we train the world model $\hat{M}$ using trajectories collected by both the goal commands from $G_{subgoals}$ (stored in D_{DAD}) and that sampled from the environment goal distribution p_g (stored in D_{egc}). Then, we sample imaginary rollouts from the world model for policy training at line 19.

Comparison with Go-Explore. We highlight the key difference between MUN's exploration strategy and the recently popular "Go-Explore" strategy (Ecoffet et al., 2019; Pislar et al., 2021; Tuyls et al., 2022; Hu et al., 2023), designed for exploration-extensive long-term GCRL settings. In Go-Explore, each training episode comprises two phases: the "Go-phase" and the "Explore-phase". During the "Go-phase," the goal-conditioned policy π_G directs the agent to an "interesting" goal (e.g., states with low frequency of occurrence in the replay buffer) (Pong et al., 2019; Pitis et al., 2020), resulting in a final state s_{T_g} after T_g steps. Following this, the "Explore-phase" begins, where an undirected exploration policy takes over from s_{T_g} for the remaining T_e timesteps. This exploration policy is trained to maximize an intrinsic exploration reward (e.g., to visit areas of the real world that the World

4

Model has not yet learned well) (Bellemare et al., 2016; Pathak et al., 2017; Burda et al., 2018; Sekar et al., 2020). This structure of training episodes has been shown to result in richer exploration (Pislar et al., 2021). In MUN, when $N_s = 2$, Algorithm 1 essentially replaces the "Explore-phase" in "Go-Explore" with another "Go-phase". Thus, the algorithm directs the agent to navigate between two "interesting" goals selected from the replay buffer. Firstly, MUN is computationally efficient as it eliminates the need to train a separate exploration policy and an ensemble of world models used to generate intrinsic exploration rewards. Secondly, MUN trains the world model for unconstrained navigation between goal states in the replay buffer, thereby improving the model's generalization to the real-world environment and leveraging the model for exploration. We empirically compare the two strategies in the context of model-based GCRL in Sec. 4.

3.2 Key Subgoal Generation through Distinct Action Discovery (DAD)

Having set up the main learning algorithm, we seek to address: how should we pick exploration-inducing goals from the replay buffer at training time to help learn generalizable world models? A straightforward strategy is to sample trajectories from the replay buffer and select subgoal states at fixed time intervals along these trajectories. We improve this simple approach with a practical method called DAD (Distinct Action Discovery) for identifying *key* subgoal states, which represent the pivotal milestones necessary to complete a complex task. Consider the block stacking task as an example. The robotic arm must be able to move its gripper to the vicinity of a block, close the gripper, lift the block, move to the top of another block, release the block, and finally open the gripper. These key subgoal states are essential for completing the task. We illustrate the roles of key states in a 3-Block Stacking task in Fig. 2(a). For an agent to learn this task, it must master reaching and navigating between the key states. By training world models for unconstrained transitions between these key states, MUN can develop models that more accurately capture the task structure and learn policies capable of adapting to novel goal scenarios e.g. block unstacking.

There exist methods for identifying key states (Zhang et al., 2021; Paul et al., 2019). However, these methods often tend to be overly complex, leading to insufficient generalization across different environments and requiring adjustments to the methods' components or parameters for various tasks. Our approach is based on

---

Algorithm 2 Key Subgoal Generation by Distinct Action Discovery

1: **Function:** DAD(...)
2: **Input:** A batch of episodes B_{egc}, number of subgoals $N_{subgoals}$
3: $A \leftarrow$ Find the most different $N_{subgoals}$ actions from B_{egc} by FPS
4: $S_{subgoals} \leftarrow$ get the corresponding states of A from B_{egc}
5: $G_{subgoals} \leftarrow \eta(S_{subgoals})$
6: **return** $G_{subgoals}$

---

the observation that certain actions are crucial at different stages of task completion. For instance, in a block stacking task, the robotic arm must learn actions such as closing the gripper, grasping the block, lifting it, and releasing the gripper. When the agent performs these key actions, the corresponding states can often be considered key subgoal states. By selecting actions that significantly differ along trajectories and extracting the corresponding states during these actions, we can identify potential key subgoal states. The Farthest Point Sampling (FPS) algorithm (Eldar et al., 1997) provides a simple and efficient method for selecting N points with maximal differences from a set. We apply FPS to choose N time steps with the greatest variations in actions from a batch of trajectory data, thereby obtaining the set of key subgoal states corresponding to these time steps. Algorithm 2 shows how MUN finds key subgoal states using the DAD method.

4 Experiments

We evaluate MUN across various robotic manipulation and navigation environments, aiming to address the following three research questions: (RQ1) Does MUN outperform other goal-conditioned model-based reinforcement learning baselines with advanced exploration strategies? (RQ2) Can DAD effectively identify key subgoal states along trajectories to the environment goal region? (RQ3) Does MUN successfully leverage the bi-directional replay buffer to train a generalizable policy for navigating effectively between arbitrary subgoals?

4.1 Environments

We conducted experiments on six challenging goal-conditioned tasks to evaluate MUN. In **Ant-Maze**, an ant-like robot is tasked to learn complex 4-legged locomotion behavior and navigate around the

5

| Ant Maze | Walker | 3-Block Stacking | Pen Rotation | Block Rotation | Fetch Slide |
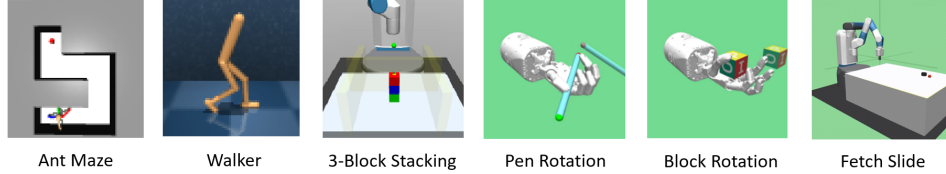
Figure 3: We evaluate MUN on 6 environments: Ant Maze, Walker, 3-Block Stacking, Block Rotation, Pen Rotation, Fetch Slide.

hallways within a maze structure. The **Walker** task involves a two-legged robot learning to control its leg joints effectively to achieve stable walking to reach goals along a flat plane forward or backward. In **3-Block Stacking**, a robot arm with a two-fingered gripper operates on a tabletop with three blocks. The goal is to stack the blocks into a tower configuration. The agent needs to learn to push, pick, and stack objects while discovering complex action sequences to complete the task in the environment. Previous solutions have relied on methods like demonstrations, curriculum learning, or extensive simulator data, highlighting the task's difficulty (Ecoffet et al., 2019; Li et al., 2020; Nair et al., 2018; Lanier, 2019). The **Block Rotation** and **Pen Rotation** tasks require the agent to manipulate a block and a pen, respectively, to achieve a randomly specified orientation along all axes. Pen Rotation is particularly challenging due to the pen's thinness, requiring precise control to prevent it from dropping. In **Fetch Slide**, a manipulator slides a puck to a designated goal area on a slippery table. Unlike tasks that involve direct manipulation, Fetch Slide emphasizes the challenge of accurately controlling the force and direction of the push operation, as the puck must slide across the flat surface to the target. See Appendix. C for more information about environments.

4.2 Baselines

We compare MUN with the following baselines. The **GC-Dreamer** baseline is discussed in Sec. 2. We include two baselines based on the Go-Explore strategy (Ecoffet et al., 2019) that has been proved efficient in the GCRL setting: MEGA (Pitis et al., 2020) and PEG (Hu et al., 2023). A Go-Explore agent firstly uses its goal-conditioned policy π^G to approach a sampled exploration-inducing goal command g, referred to as the Go-phase. In the Explore-phase, it activates an exploration policy π^E to explore the environment from the terminal state of the Go-phase. In contrast, MUN improves the generalization of world models to facilitate effective real-world environment exploration. During training, MUN collects trajectories that navigate between two goal states sampled from its candidate subgoal set, essentially replacing the "Explore-phase" in "Go-Explore" with another "Go-phase". MEGA commands the agent to rarely seen states at the frontier by using kernel density estimates (KDE) of state densities and chooses low-density goals from the replay buffer. PEG selects goal commands to guide an agent's goal-conditioned policy toward states with the highest exploration potential given its current level of training. This potential is defined as the expected accumulated exploration reward during the Explore-phase. Similar to MUN, our baseline methods, named **PEG-G** and **MEGA-G**, augment **GC-Dreamer** with the PEG and MEGA Go-Explore strategies, respectively. In these methods, the replay buffer D contains not only trajectories sampled by the GCRL policy π_G commanded by environment goals but also exploratory trajectories sampled using the Go-Explore strategies. The exploration policy π^E in **PEG-G** and **MEGA-G** is the Plan2Explore policy from Sekar et al. (2020), which encourages the agent to actively search for states that induce disparities among an ensemble of world models.

We note that MUN and the baselines are all implemented based on the Dreamer framework as realized in **GC-Dreamer**[1].

4.3 Results

Fig. 4 shows the evaluation performance of MUN and all baselines across training. MUN demonstrates superior performance compared to the baseline models, excelling in both the final success rate and the speed of learning. MUN outperforms the Go-Explore baselines (MEGA-G and PEG-G) across all tasks, demonstrating the effectiveness of the exploration strategy in MUN over the alternative Go-Explore strategies. In the most challenging tasks—block stacking, block rotation, and pen rotation—MUN shows a significant margin of superiority. For example, MUN achieves over 95%

---

[1]MUN is not tied to a specific world model architecture and can be applied to any model-based RL framework.

6

(a) Ant Maze   (b) Walker   (c) 3-Block Stacking

(d) Block Rotation   (e) Pen Rotation   (f) Fetch Slide
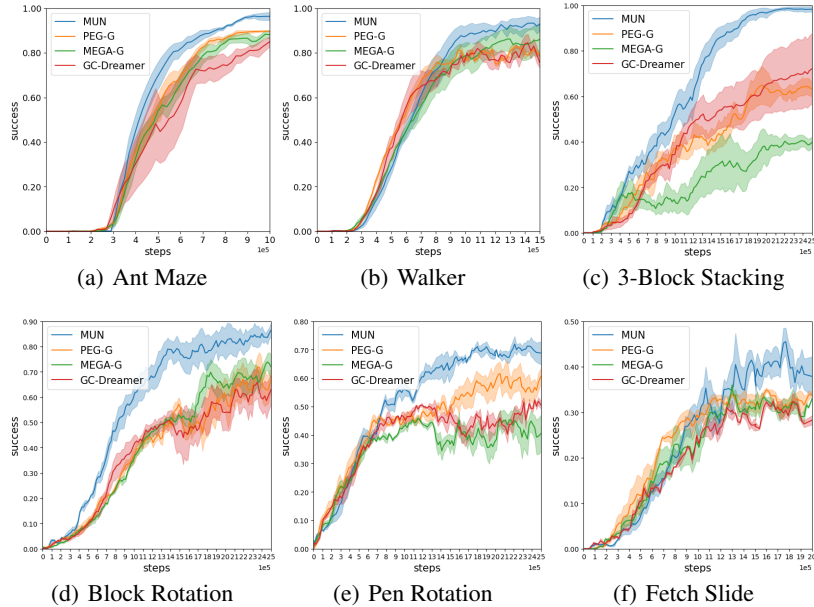
Figure 4: Experiment results comparing MUN with the baselines over 5 random seeds.

success rate on 3 block stacking, while all other baselines only manage to achieve around 60% success rate on this task within 2.5M steps. MEGA-G and PEG-G heuristically pick exploration-inducing goals to initiate exploration by a separate policy. Since finding a goal state that is optimally aligned with both the goal-conditioned policy and the exploration policy is challenging, these methods can result in suboptimal goals, thereby slowing down exploration. GC-Dreamer lacks a Go-Explore phase, which limits its exploration potential. Despite this, it can still perform comparably to or even better than MEGA-G and PEG-G in certain contexts. This indicates that the Go-Explore strategy does not always guarantee improved exploration, and suboptimal goal-setting during the "Go-phase" can hinder exploration (see 3 block stacking).

Fetch Slide is a non-prehensile manipulation task. This environment has asymmetric state transitions: when the puck is slid outside the robot's workspace, the manipulator cannot reach the puck's position to slide it backward due to physical constraints. MUN still outperforms the other baselines in this environment. We found MUN, with the DAD strategy, can discover key subgoals for this task, like contacting the puck, placing the manipulator at different angles around the puck, and stabilizing the manipulator upon reaching the goal (these key states result from distinct actions). MUN enables learning state transitions between these key subgoals to discover the high-level task structure.

(a) 3-Block Stacking   (b) Pen Rotation

Figure 5: The world model prediction error curves throughout the training steps for 3-Block Stacking and Pen Rotation.

It learns a generalizable world model that handles sliding the puck between any positions within the workspace and predicts low probabilities for infeasible transitions from puck positions outside the workspace. Particularly, it enables the agent to hit the puck multiple times if it is within its workspace, thereby improving task success rates. That said, the current goal selection mechanism in MUN lacks a process to filter out infeasible goals from the current state, which could adversely affect sample efficiency. We left addressing this limitation and implementing a robust filtering mechanism for infeasible goals as a focus for future work.

We studied the prediction error of learned world models in MUN and the baselines. Fig. 5 shows the one-step model prediction error throughout the training steps. The world models trained by MUN

7

show a much smaller generalization gap to the real environment compared to the baselines across the training steps. Consequently, MUN can effectively leverage these higher-quality world models to train policies that generalize better to the real environment. We present a quantitative comparison of the world model prediction quality between MUN and the baselines in terms of model prediction *compounding error* in Appendix F.3.

4.4 Can DAD find key subgoals?
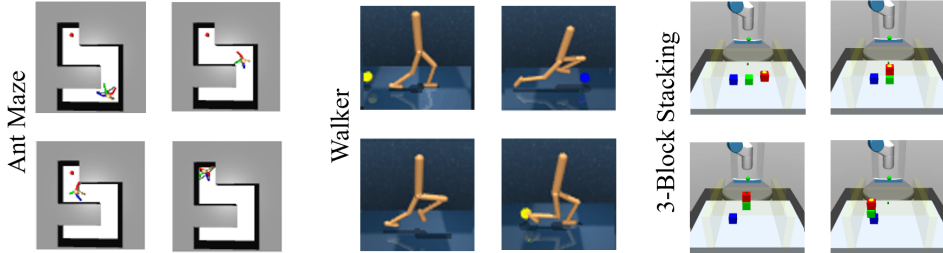
Figure 6: Key subgoals found by DAD (Algorithm 2) in three environments: Ant-Maze, Walker, 3-Block Stacking. They present the important landmarks on the path to the task goal regions.

We visualize several subgoals found by the DAD algorithm during the training process in Fig. 6 for three environments: Ant-Maze, Walker, 3-Block Stacking. In **Walker**, DAD successfully identifies the crucial joint angles and forces of the Walker robot during its forward locomotion, including standing, striding, jumping, landing, and leg support. In **Ant-Maze**, DAD recognizes significant motion variations at corridor corners. In **3-Block Stacking**, DAD successfully identifies crucial state transitions required during the stacking process. These critical subgoals include block grasping, lifting, horizontal movement, vertical movement, and gripper release. For more discussion about subgoals found by the DAD in other environments, please refer to Appendix F.1.

4.5 Can MUN navigate between arbitrary subgoals?
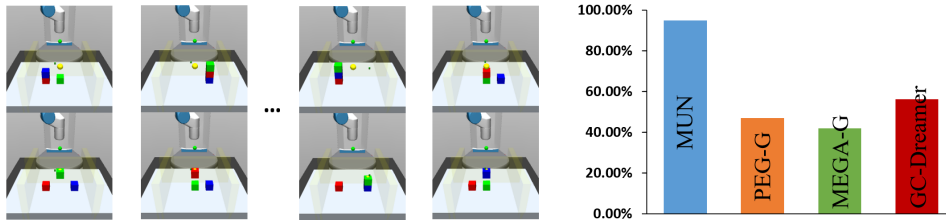
Figure 7: Experiment setup and results of navigation between any pair of subgoals in the 3-Block Stacking environment. In the left part, the bottom section of each image depicts the ultimate evaluation goal for one evaluation episode, while the top section illustrates the manually set initial state. The right part shows the evaluation success rates.

As MUN is capable of identifying pivotal subgoal states necessary for complex tasks and training world models and policies for seamless transitions between these subgoals, we investigate MUN's capacity to generalize to new task settings concerning important subgoals. We set the initial state of the agent at one random subgoal and command it to reach another random subgoal. Such task setting is *not provided to the agent during training*. For the 3 Block Stacking task, we employ a set of 15 manually created subgoals representing various critical states in the block-stacking process, resulting in 225 unique combinations of initial states and test goals for evaluation. Each combination undergoes 10 repeated evaluations, totaling 2250 evaluation trajectories. These evaluations encompass both the forward and reverse processes of stacking and unstacking blocks, assessing the agent's proficiency in both task completion and restoration. For example, in the left portion of Fig. 7, we visualize some subgoals used as initial task state in the upper part and some subgoals used as evaluation test goals in the lower part. The right section of Fig. 7 illustrates MUN's superiority over the other baselines in these evaluation experiments, achieving the highest success rate through its ability to develop

8

<div align="center">(a) 3-Block Stacking   (b) Block Rotation   (c) Pen Rotation</div>
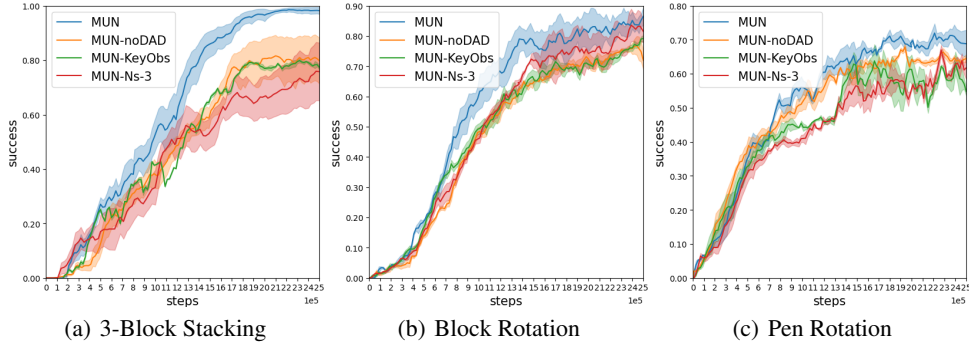
Figure 8: Experiment results comparing MUN with its ablations over 5 random seeds.

a robust and adaptable world model that generalizes to novel tasks. Additional results in different environments are provided in Appendix F.2.

4.6 Ablation study

We conducted the following ablation studies to investigate MUN's exploration goal selection mechanism. First, we investigated the effect of the number of subgoal states (N_s) in our algorithm. MUN sequentially traverses $N_s = 2$ goal states sampled from the replay buffer to explore the environment during each training episode. We introduced an ablation **MUN-Ns-3** that sets $N_s = 3$. This ablation aims to investigate whether increasing N_s leads to improved learning performance. Second, we considered an ablated version of MUN, named **MUN-noDAD**, which replaces the goal sampling strategy DAD (Algorithm 2) with a simple method that chooses goal states with fixed time interval in trajectories sampled from the replay buffer. This ablation investigates the importance of identifying key subgoal states, which represent pivotal milestones necessary to complete a complex task. It seeks to determine whether training world models from state transitions between these key states in MUN is essential, or if using any states from the replay buffer would suffice. Lastly, we explored an alternative key subgoal discovery strategy. MUN identifies key subgoals for exploration as states in the replay buffer that result in distinct actions within the action space. We introduced an ablation, **MUN-KeyObs**, which directly discovers key subgoals from the state space by identifying centroids of (latent) state clusters in the replay buffer, following the strategy in Zhang et al. (2021).

The results are depicted in Fig. 8. MUN outperforms all ablated versions. Setting $N_s = 3$ slows down the training performance, supporting our claim it suffices to set $N_s = 2$. The performance of MUN-noDAD and MUN-KeyObs does not match MUN, especially in the 3 Block Stacking environment, highlighting that discovering key subgoals in the action space (the DAD strategy) indeed contributes to higher performance and efficiency. It is noteworthy that the ablation methods achieve a relatively small gap in success rates compared to MUN in the challenging Block Rotation and Pen Rotation environments. This suggests that MUN's approach to learning a world model from state transitions between any states in the replay buffer (whether tracing back along recorded trajectories or transitioning across separate trajectories) alone is effective in bridging the generalization gap between the model and the real environment.

5 Related Work

Model-based reinforcement learning (MBRL) is a promising approach to reinforcement learning that learns a model of the environment and uses it to plan actions (Sutton, 1991; Deisenroth and Rasmussen, 2011; Oh et al., 2017; Chua et al., 2018). It has achieved remarkable success in numerous control tasks and games, such as chess (Silver et al., 2017; Schrittwieser et al., 2020; Xu et al., 2022), Atari games (Hafner et al., 2020; Schrittwieser et al., 2020; Oh et al., 2017), continuous control tasks (Kurutach et al., 2018; Buckman et al., 2018; Hafner et al., 2019b; Janner et al., 2019), and robotic manipulation tasks (Lowrey et al., 2018; Luo et al., 2018). The dynamic model serves as a pivotal component of model-based reinforcement learning, primarily fulfilling two key roles: planning actions (Deisenroth and Rasmussen, 2011; Oh et al., 2017; Chua et al., 2018; Lowrey et al., 2018; Hafner et al., 2019b) or generating synthetic data to aid in the training of model-free reinforcement

9

learning algorithms (Janner et al., 2019; Hafner et al., 2020, 2023). The primary drawback of the former lies in the excessive cost associated with long-term planning. To address this issue, the concept of ensemble has been employed to enhance performance (Chua et al., 2018; Kurutach et al., 2018; Buckman et al., 2018). Oh et al. (2017) and Hansen et al. (2022b) integrate the dynamics model with a value prediction network to improve the accuracy of long-term planning. The latter also suffers from the potential bias of the model, which can result in inaccuracies in the generated data, thereby directly impacting policy learning (Luo et al., 2018; Lai et al., 2021).

Multi-goal reinforcement learning (RL) agents (Schaul et al., 2015; Plappert et al., 2018; Ghosh et al., 2019) acquire goal-conditioned behaviors capable of achieving and generalizing across diverse sets of objectives. Researchers have been continuously exploring the integration of Model-based RL and Goal-conditioned RL (Mendonca et al., 2021; Nair et al., 2020; Zhang et al., 2020), leveraging the capabilities of dynamic models in planning and generating synthetic data to enhance the training efficiency and generalization of GCRL. However, compared to traditional RL problems, GCRL faces more severe challenges regarding reward sparsity and exploration difficulties (Ren et al., 2019; Florensa et al., 2018; Trott et al., 2019). These challenges often lead to significant biases in the learned World Model, consequently impairing the performance of goal-conditioned policies (Mendonca et al., 2021; Hu et al., 2023). Pong et al. (2019) propose to learn a maximum-entropy goal distribution, Pitis et al. (2020) encourage the agent to explore goals with low frequency of occurrence in the replay buffer. Sekar et al. (2020) introduce a planning algorithm to pick goals for exploration using World Models.

World Models hold significant promise for GCRL, as they enable fast exploration and support the training of more generalized policies (McCarthy et al., 2021; Shyam et al., 2019; Hu et al., 2023; Sekar et al., 2020). Within this framework, learning a reliable World Model is essential for developing effective policies (Zhang et al., 2024; Young et al., 2022; Wang et al., 2023; Lai et al., 2021). Kauvar et al. (2023) propose a curiosity-driven exploration method, which is focused on replay buffer management. Hansen et al. (2022a) use demonstration data as a supplement to the replay buffer to learn a more reliable World Model. Previous work has often focused on devising more appropriate objectives when sampling real trajectory data from the environment to enrich the diversity of dynamic transitions in the replay buffer (Nair et al., 2020; Charlesworth and Montana, 2020; Trott et al., 2019; Florensa et al., 2018; Campero et al., 2020). However, they overlooked the overall direction of dynamic transitions within the data which extremely affects the richness of dynamic transitions to learn a comprehensive World Model.

6 Conclusion

In summary, we introduce MUN, a novel goal-directed exploration algorithm designed for effective world modeling of seamless transitions between arbitrary states in replay buffers, whether retracing along recorded trajectories or transitioning between states on separate trajectories. As the quality of the world model improves, MUN demonstrates high efficacy in learning goal-conditioned policies in sparse-reward environments. Additionally, we present a practical strategy DAD for identifying pivotal subgoal states, which act as critical milestones in completing complex tasks. The experimental results underscored the effectiveness of MUN in strengthening the reliability of world models and learning policies capable of adapting to novel test goals.

Reproducibility Statement

The code for MUN is available on `https://github.com/RU-Automated-Reasoning-Group/MUN`. For hyperparameter settings and baseline pseudocode, please refer to Appendix D and Appendix E.3.

Acknowledgements

We thank the anonymous reviewers for their comments and suggestions. This work was supported by NSF Award #CCF-2124155.

10

References

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Pieter Abbeel, O., and Zaremba, W. (2017). Hindsight experience replay. *Advances in neural information processing systems*, 30.

Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. (2016). Unifying count-based exploration and intrinsic motivation. *Advances in neural information processing systems*, 29.

Buckman, J., Hafner, D., Tucker, G., Brevdo, E., and Lee, H. (2018). Sample-efficient reinforcement learning with stochastic ensemble value expansion. *Advances in neural information processing systems*, 31.

Burda, Y., Edwards, H., Storkey, A., and Klimov, O. (2018). Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*.

Campero, A., Raileanu, R., Küttler, H., Tenenbaum, J. B., Rocktäschel, T., and Grefenstette, E. (2020). Learning with amigo: Adversarially motivated intrinsic goals. *arXiv preprint arXiv:2006.12122*.

Charlesworth, H. and Montana, G. (2020). Plangan: Model-based planning with sparse rewards and multiple goals. *Advances in Neural Information Processing Systems*, 33:8532–8542.

Chua, K., Calandra, R., McAllister, R., and Levine, S. (2018). Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *Advances in neural information processing systems*, 31.

Deisenroth, M. and Rasmussen, C. E. (2011). Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472.

Ecoffet, A., Huizinga, J., Lehman, J., Stanley, K. O., and Clune, J. (2019). Go-explore: a new approach for hard-exploration problems. *arXiv preprint arXiv:1901.10995*.

Eldar, Y., Lindenbaum, M., Porat, M., and Zeevi, Y. Y. (1997). The farthest point strategy for progressive image sampling. *IEEE transactions on image processing*, 6(9):1305–1315.

Florensa, C., Held, D., Geng, X., and Abbeel, P. (2018). Automatic goal generation for reinforcement learning agents. In *International conference on machine learning*, pages 1515–1528. PMLR.

Georgiev, I., Giridhar, V., Hansen, N., and Garg, A. (2024). Pwm: Policy learning with large world models. *arXiv preprint arXiv:2407.02466*.

Ghosh, D., Gupta, A., Reddy, A., Fu, J., Devin, C., Eysenbach, B., and Levine, S. (2019). Learning to reach goals via iterated supervised learning. *arXiv preprint arXiv:1912.06088*.

Hafner, D., Lillicrap, T., Ba, J., and Norouzi, M. (2019a). Dream to control: Learning behaviors by latent imagination. *arXiv preprint arXiv:1912.01603*.

Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., and Davidson, J. (2019b). Learning latent dynamics for planning from pixels. In *International conference on machine learning*, pages 2555–2565. PMLR.

Hafner, D., Lillicrap, T., Norouzi, M., and Ba, J. (2020). Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*.

Hafner, D., Pasukonis, J., Ba, J., and Lillicrap, T. (2023). Mastering diverse domains through world models. *arXiv preprint arXiv:2301.04104*.

Hansen, N., Lin, Y., Su, H., Wang, X., Kumar, V., and Rajeswaran, A. (2022a). Modem: Accelerating visual model-based reinforcement learning with demonstrations. *arXiv preprint arXiv:2212.05698*.

Hansen, N., Su, H., and Wang, X. (2023). Td-mpc2: Scalable, robust world models for continuous control. *arXiv preprint arXiv:2310.16828*.

11

Hansen, N., Wang, X., and Su, H. (2022b). Temporal difference learning for model predictive control. *arXiv preprint arXiv:2203.04955*.

Hu, E. S., Chang, R., Rybkin, O., and Jayaraman, D. (2023). Planning goals for exploration. *arXiv preprint arXiv:2303.13002*.

Janner, M., Fu, J., Zhang, M., and Levine, S. (2019). When to trust your model: Model-based policy optimization. *Advances in neural information processing systems*, 32.

Kauvar, I., Doyle, C., Zhou, L., and Haber, N. (2023). Curious replay for model-based adaptation. *arXiv preprint arXiv:2306.15934*.

Kurutach, T., Clavera, I., Duan, Y., Tamar, A., and Abbeel, P. (2018). Model-ensemble trust-region policy optimization. *arXiv preprint arXiv:1802.10592*.

Lai, H., Shen, J., Zhang, W., Huang, Y., Zhang, X., Tang, R., Yu, Y., and Li, Z. (2021). On effective scheduling of model-based reinforcement learning. *Advances in Neural Information Processing Systems*, 34:3694–3705.

Lai, Y., Wang, W., Yang, Y., Zhu, J., and Kuang, M. (2020). Hindsight planner. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, pages 690–698.

Lanier, J. B. (2019). *Curiosity-driven multi-criteria hindsight experience replay*. University of California, Irvine.

Li, R., Jabri, A., Darrell, T., and Agrawal, P. (2020). Towards practical multi-object manipulation using relational reinforcement learning. In *2020 ieee international conference on robotics and automation (icra)*, pages 4051–4058. IEEE.

Lowrey, K., Rajeswaran, A., Kakade, S., Todorov, E., and Mordatch, I. (2018). Plan online, learn offline: Efficient learning and exploration via model-based control. *arXiv preprint arXiv:1811.01848*.

Luo, Y., Xu, H., Li, Y., Tian, Y., Darrell, T., and Ma, T. (2018). Algorithmic framework for model-based deep reinforcement learning with theoretical guarantees. *arXiv preprint arXiv:1807.03858*.

McCarthy, R., Wang, Q., and Redmond, S. J. (2021). Imaginary hindsight experience replay: Curious model-based learning for sparse reward tasks. *arXiv preprint arXiv:2110.02414*.

Mendonca, R., Rybkin, O., Daniilidis, K., Hafner, D., and Pathak, D. (2021). Discovering and achieving goals via world models. *Advances in Neural Information Processing Systems*, 34:24379–24391.

Nagabandi, A., Konolige, K., Levine, S., and Kumar, V. (2020). Deep dynamics models for learning dexterous manipulation. In *Conference on Robot Learning*, pages 1101–1112. PMLR.

Nair, A., McGrew, B., Andrychowicz, M., Zaremba, W., and Abbeel, P. (2018). Overcoming exploration in reinforcement learning with demonstrations. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 6292–6299. IEEE.

Nair, S., Savarese, S., and Finn, C. (2020). Goal-aware prediction: Learning to model what matters. In *International Conference on Machine Learning*, pages 7207–7219. PMLR.

Oh, J., Singh, S., and Lee, H. (2017). Value prediction network. *Advances in neural information processing systems*, 30.

Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. (2017). Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*, pages 2778–2787. PMLR.

Paul, S., Vanbaar, J., and Roy-Chowdhury, A. (2019). Learning from trajectories via subgoal discovery. *Advances in Neural Information Processing Systems*, 32.

Pislar, M., Szepesvari, D., Ostrovski, G., Borsa, D., and Schaul, T. (2021). When should agents explore? *arXiv preprint arXiv:2108.11811*.

12

Pitis, S., Chan, H., Zhao, S., Stadie, B., and Ba, J. (2020). Maximum entropy gain exploration for long horizon multi-goal reinforcement learning. In *International Conference on Machine Learning*, pages 7750–7761. PMLR.

Plappert, M., Andrychowicz, M., Ray, A., McGrew, B., Baker, B., Powell, G., Schneider, J., Tobin, J., Chociej, M., Welinder, P., et al. (2018). Multi-goal reinforcement learning: Challenging robotics environments and request for research. *arXiv preprint arXiv:1802.09464*.

Pong, V. H., Dalal, M., Lin, S., Nair, A., Bahl, S., and Levine, S. (2019). Skew-fit: State-covering self-supervised reinforcement learning. *arXiv preprint arXiv:1903.03698*.

Ren, Z., Dong, K., Zhou, Y., Liu, Q., and Peng, J. (2019). Exploration via hindsight goal generation. *Advances in Neural Information Processing Systems*, 32.

Schaul, T., Horgan, D., Gregor, K., and Silver, D. (2015). Universal value function approximators. In *International conference on machine learning*, pages 1312–1320. PMLR.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., et al. (2020). Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609.

Sekar, R., Rybkin, O., Daniilidis, K., Abbeel, P., Hafner, D., and Pathak, D. (2020). Planning to explore via self-supervised world models. In *International conference on machine learning*, pages 8583–8592. PMLR.

Shyam, P., Jaśkowski, W., and Gomez, F. (2019). Model-based active exploration. In *International conference on machine learning*, pages 5779–5788. PMLR.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. (2017). Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*.

Sutton, R. S. (1991). Dyna, an integrated architecture for learning, planning, and reacting. *ACM Sigart Bulletin*, 2(4):160–163.

Trott, A., Zheng, S., Xiong, C., and Socher, R. (2019). Keeping your distance: Solving sparse reward tasks using self-balancing shaped rewards. *Advances in Neural Information Processing Systems*, 32.

Tuyls, J., Yao, S., Kakade, S., and Narasimhan, K. (2022). Multi-stage episodic control for strategic exploration in text games. *arXiv preprint arXiv:2201.01251*.

Wang, X., Wongkamjan, W., Jia, R., and Huang, F. (2023). Live in the moment: Learning dynamics model adapted to evolving policy. In *International Conference on Machine Learning*, pages 36470–36493. PMLR.

Williams, G., Aldrich, A., and Theodorou, E. (2015). Model predictive path integral control using covariance variable importance sampling. *arXiv preprint arXiv:1509.01149*.

Xu, Y., Hansen, N., Wang, Z., Chan, Y.-C., Su, H., and Tu, Z. (2022). On the feasibility of cross-task transfer with model-based reinforcement learning. *arXiv preprint arXiv:2210.10763*.

Young, K., Ramesh, A., Kirsch, L., and Schmidhuber, J. (2022). The benefits of model-based generalization in reinforcement learning. *arXiv preprint arXiv:2211.02222*.

Yu, T., Thomas, G., Yu, L., Ermon, S., Zou, J. Y., Levine, S., Finn, C., and Ma, T. (2020). Mopo: Model-based offline policy optimization. *Advances in Neural Information Processing Systems*, 33:14129–14142.

Zhang, L., Yang, G., and Stadie, B. C. (2021). World model as a graph: Learning latent landmarks for planning. In *International conference on machine learning*, pages 12611–12620. PMLR.

Zhang, W., Wang, G., Sun, J., Yuan, Y., and Huang, G. (2024). Storm: Efficient stochastic transformer based world models for reinforcement learning. *Advances in Neural Information Processing Systems*, 36.

Zhang, Y., Abbeel, P., and Pinto, L. (2020). Automatic curriculum learning through value disagreement. *Advances in Neural Information Processing Systems*, 33:7648–7659.

13

Appendix

A Extended Background

A.1 Dreamer World Model

The RSSM consists of an encoder, a recurrent model, a representation model, a transition predictor, and a decoder, as formulated in Equation 3. And it employs an end-to-end training methodology, where its parameters are jointly optimized based on the loss functions of various components, including dynamic transition prediction, reward prediction, and observation encoding-decoding. These components often operate in a latent space rather than the original observation space, as encoded by the World Model. Therefore, during end-to-end training, the losses of all components indirectly optimize the latent space.

The encoder f_E encodes the input state x_t into a embed state e_t, which is then fed with the deterministic state h_t into the representation model q_ϕ to generate the posterior state z_t. The transition predictor p_ϕ predicts the prior state $\hat{z}_t$ based on the deterministic state h_t without access to the current input state x_t. Using the concatenation of either (h_t, z_t) or $(h_t, \hat{z}_t)$ as input, the recurrent transition function $f\phi$ iteratively updates the deterministic state h_t with given action a_t.

$$
\begin{aligned}
\text{Encoder:} \quad & e_t = f_E(e_t|x_t) \\
\text{Recurrent model:} \quad & h_t = f_\phi(h_{t-1}, z_{t-1}, a_{t-1}) \\
\text{Representation model:} \quad & z_t \sim q_\phi(z_t|h_t, e_t) \\
\text{Transition predictor:} \quad & \hat{z}_t \sim p_\phi(\hat{z}_t|h_t) \\
\text{Decoder:} \quad & \hat{x}_t \sim f_D(\hat{x}_t|h_t, z_t)
\end{aligned}
\tag{3}
$$

A.2 Temporal Distance Training in LEXA

The goal-reaching reward r^G is defined by the self-supervised temporal distance objective (Mendonca et al., 2021) which aims to minimize the number of action steps needed to transition from the current state to a goal state within imagined rollouts. We use b_t to denote the concatenate of the deterministic state h_t and the posterior state z_t at time step t.

$$
b_t = (h_t, z_t)
\tag{4}
$$

The temporal distance D_t is trained by sampling pairs of imagined states b_t, b_{t+k} from imagined rollouts and predicting the action steps number k between the embedding of them, with a predicted embedding $\hat{e}_t$ from b_t to approximate the true embedding e_t of the observation x_t.

$$
\text{Predicted embedding:} \quad emb(b_t) = \hat{e}_t \approx e_t, \quad \text{where} \quad e_t = f_E(x_t)
\tag{5}
$$

$$
\text{Temporal distance: } D_t(\hat{e}_t, \hat{e}_{t+k}) \approx k/H \quad \text{where} \quad \hat{e}_t = emb(b_t) \quad \hat{e}_{t+k} = emb(b_{t+k})
\tag{6}
$$

$$
r_t^G(b_t, b_{t+k}) = -D_t(\hat{e}_t, \hat{e}_{t+k})
\tag{7}
$$

B Limitations and Future Work

The MUN has provided powerful guidance in enhancing world model learning by repeatedly studying the transitions between various key states. This allows the acquisition of richer dynamic transitions and deepens the world model's understanding of the real world. However, such a framework requires an efficient strategy for discovering key states, as evidenced by the comparative results of the MUN and MUN-noDAD. We found that although DAD excels in discovering key states with its simple and efficient method, it will identify ineffective and task-irrelevant states in tasks with highly complex

14

action spaces or weak correlations between goal space and action space. This can lead to the degradation of the MUN architecture due to poor-quality subgoals, resulting in a substantial amount of ineffective sampling in the environment. Therefore, for environments with more complex action and goal spaces, we need to develop a more robust and effective method for discovering subgoals than DAD. Only with an efficient and powerful self-supervised subgoal discovery mechanism can the MUN framework be fully utilized.

Meanwhile, MUN autonomously discovers subgoals and learns a more robust and comprehensive world model by randomly navigating between subgoals. Although the MUN has achieved huge success in model-based reinforcement learning (MBRL), we believe it can also be applied to general model-free methods. General model-free methods do not require learning a world model and have a simpler architecture. The MUN can directly guide the goal-conditioned policy to enhance learning in navigation between different subgoals. It can use sampled trajectories to learn this policy directly, bypassing the use of the world model to train policies and value functions through simulated trajectories, thereby enhancing the agent's ability to reach unconstrained goals. Therefore, we plan to explore the application and effectiveness of the MUN in model-free RL in the future and develop a new robust self-supervised subgoal discovery mechanism to make the MUN applicable to more complex environments.

C   Environments

C.1   3-Block Stacking

In this task, the robot must stack three blocks in different colors into a tower shape. While PEG assesses goals of varying difficulty levels: 3 easy goals (picking up a single block), 6 medium goals (stacking two blocks), and 6 hard goals (stacking three blocks), our evaluation is focused solely on the 6 hard goals, and we use only 3 hard goals of them as the guiding goals from the training environment. Training and evaluating with only the hardest goals imposes a significant challenge for the MUN. However, we observed that the MUN can spontaneously discover additional easy and medium goals through DAD, as these serve as critical transitional states toward the hard goals. The environment is characterized by a 14-dimensional state and goal space. The first five dimensions capture the gripper's state, while the remaining nine dimensions correspond to the xyz positions of each block. The action space is 4-dimensional, with three dimensions dedicated to the gripper's xyz movements and the fourth dimension controlling the gripper's finger movement. Success is defined by achieving an L2 distance of less than 3 cm between each block's xyz position and its target position. This environment is a modified version of the FetchStack3 environment from Pitis et al. (2020), designed to better test the robot's precision in stacking.

C.2   Walker

In this environment, a 2D walker robot is trained and evaluated on its ability to move across a flat surface. The environment's implementation is based on the code from Mendonca et al. (2021). To thoroughly assess the agent's capability and precision in covering longer distances, we expanded the evaluation goals to 12 ($\pm 13, \pm 16, \pm 19, \pm 22, \pm 25, \pm 28$) along the x axis from the initial position. In our training setup for the MUN, we only use the goals at ± 13 and ± 16 provided by the environment, but we evaluate the agent's performance across all 12 goals. Success is measured by verifying whether the agent's x position is within a small margin of the target x position. The state and goal space in this environment are nine-dimensional, comprising the walker's xz positions and its joint angles. This configuration ensures a comprehensive evaluation of the walker's locomotion capabilities.

C.3   Ant Maze

This environment builds upon the Ant Maze from Pitis et al. (2020), incorporating a few modifications. The state and goal spaces in the Ant Maze environment are highly complex, totaling 29 dimensions. These dimensions include the ant's xyz position, joint angles, and velocities. The first three dimensions account for the xyz position, the next 12 dimensions capture the joint angles of the ant's limbs, and the remaining 14 dimensions represent the velocities of the joints and the ant's movements in the xy plane. The action space consists of 8 dimensions, controlling the hip and ankle movements of the ant's legs.We matched the goal space to the state space, which includes the ant's

15

xyz coordinates, joint positions, and velocities. We also introduced an additional room in the top left to increase the difficulty like PEG. In this scenario, the ant robot must traverse from the bottom left to the top left of a maze, navigating through various corridors. The task is particularly challenging due to its lengthy duration—each episode lasts 500 timesteps—and the significant distance the ant must cover. Unlike PEG, which evaluates goals in both the central hallway and the top left room, our evaluation focuses exclusively on the four most difficult goals located in the top left room. For training, we utilize all 32 goals throughout the maze. The maze itself measures about 6 by 8 meters. The ant succeeds if its xy position is within 1.0 meter of the goal, roughly the size of a single cell in the maze.

C.4   Fetch Slide

In this task, a robotic arm with a two-fingered gripper must push an object along a flat surface to a specific goal location. We use the "FetchSlide-v1" environment from Gymnasium, where the robot operates in a 25-dimensional state space that includes the robot's joint states, object position, and goal information. The goal space is 3-dimensional, representing the target coordinates for the object. Each episode presents a unique random goal location within a bounded area, requiring the agent to adjust its pushing strategy accordingly. A key challenge in Fetch Slide lies in the indirect manipulation of the object. The agent must accurately control the force and direction of its push while accounting for physical properties like friction, surface irregularities, and object momentum. Unlike grasping or lifting tasks, sliding demands precise force calibration and anticipation of the object's response to contact. For evaluation, the agent's learned policy is tested across 50 episodes with different goal locations, assessing its ability to generalize over varied configurations. Training goals are randomly generated from the environment, helping the agent explore diverse sliding trajectories to improve robustness across different scenarios.

C.5   Block and Pen Rotation

In this task, a robotic hand must manipulate either a thin pen or a block to achieve specified target rotations. We use "HandManipulatePenRotate-v1" and "HandManipulateBlockRotateXYZ-v1" versions of the gymnasium environments. Both tasks feature a state space of 61 dimensions, encompassing the robot's joint states, object states, and goal information. The goal space is 7-dimensional, representing the target pose details. Each episode will have randomized target rotations goal for all axes of the block and for the x and y axes of the pen. The pen is more challenging to handle due to its tendency to slip, requiring more precise control compared to the block. For evaluation, the latest policy is tested 50 episodes for each task, with each episode having a unique random goal. In our framework, training goals are also randomly generated from the environment.

D   Baselines

We first present our overall training framework for goal-conditioned model-based reinforcement learning (MBRL). It is important to note that all baselines utilize this training framework, differing only in the strategy employed for collecting trajectories within the real environment. Our training framework is based on the implementation of LEXA (Mendonca et al., 2021).

Algorithm 3 General MBRL Training Framework

1: **Input:** Policy π^G, π^E, Environment Goal Distribution G, World Model $\hat{M}$, reward function r^G, r^E
2: $\mathcal{D} \leftarrow \{\}$ Initialize buffer.
3: **for** Episode $i = 1$ to N_{train} **do**
4:     $\tau \leftarrow$ Collect trajectories$(\dots)$
5:     $\mathcal{D} \leftarrow \mathcal{D} \cup \tau$
6:     Update world model $\hat{M}$ with $\mathcal{D}$
7:     Update π^G in imagination with $\hat{M}$ to maximize r^G
8:     Update π^E in imagination with $\hat{M}$ to maximize r^E

16

D.1 Go-Explore

Our baselines utilize the state-of-the-art Go-Explore exploration framework, following the implementation detailed in PEG (Hu et al., 2023). This approach initially employs a goal-conditioned policy π^G to get as close as possible to a specified goal g, a process referred to as the "Go phase." Subsequently, an explorer policy π^E is used to further explore the environment starting from the final state of the Go phase, known as the "Explore phase."

The quality of the trajectories generated by the Go-Explore strategy largely depends on the selection of the goal g during the Go phase. Therefore, establishing an effective mechanism for selecting the Go phase goals is crucial. If the chosen goal g is too simple, the explorer will not sufficiently explore the environment. Conversely, if the goal g is too difficult, the goal-achieving policy π^G will fail to approach it effectively. Thus, the baselines MEGA-G and PEG-G employ different goal selection strategies to determine g, guiding the agent to areas with high exploration potential during the Go phase. MEGA-G and PEG-G enhance the agent's exploration efficiency by crafting robust exploration strategies, enabling faster learning of the world model with respect to new dynamic transitions and environmental areas. We present the pseudocode for Go-Explore in Algorithm 4.

Algorithm 4 Go Explore Framework

1: **function** GO-EXPLORE(g, π^G, π^E)
2:     $s_0 \leftarrow$ env.reset()
3:     $\tau \leftarrow \{s_0\}$
4:     **for** Step $t = 1$ to T_{Go} **do**
5:         $s_t \leftarrow$ env.step($\pi^G(s_{t-1}, g)$)
6:         $\tau \leftarrow \tau \cup \{s_t\}$
7:         **if** agent reach g **then**
8:             break
9:     $t_e = t$
10:     **for** Step $t = t_e$ to $t_e + T_{\text{Explore}}$ **do**
11:         $s_t \leftarrow$ env.step($\pi^E(s_{t-1})$)
12:         $\tau \leftarrow \tau \cup \{s_t\}$
13: **return** τ

D.2 GC-Dreamer

GC-Dreamer is the goal-conditioned version of Dreamer (Hafner et al., 2019a,b, 2020), without incorporating any exploration or goal-directed strategies. It only uses a goal-conditioned policy to collect trajectories, with goals provided by the training environment.

Algorithm 5 GC-Dreamer Goal Sampling

1: **function** COLLECT TRAJECTORIES(. . .)
2:     $g \leftarrow$ Returned by environment
3:     $\tau \leftarrow$ Sample a trajectories by π^G using goal g
4: **return** τ

D.3 PEG-G

PEG uses a world model to simulate exploration trajectories and evaluates the exploration potential($P^E(g)$) to identify areas worth exploring.

$$P^E(g) = \mathbb{E}_{p_{\pi^G(\cdot|\cdot,g)(s_T)}}[V^E(s_T)] \tag{8}$$

$$V^E(s_T) = \mathbb{E}_{\pi^E}\left[\sum_{t=T+1}^{T+T_E} \gamma^{t-T-1} r_t^E \right] \tag{9}$$

17

PEG set goal g for the goal-conditioned policy and generalize it to K trajectories using world model. s_T denotes the final state of the goal-conditioned trajectory from the "Go phase" of Go-Explore to reach the g. Since the objective in Equation 8 is not easily computable, as it relies on the final state distribution induced by the target-conditioned policy π^g, which may rapidly change throughout the training process, it's crucial to use the latest estimates for better exploration. PEG achieve this by leveraging the learned world model. PEG utilize the learned exploration value function $V_E(s_k^T)$ (Equation 10) from the learned world model to estimate the exploration value of the final state for each trajectory, and average these estimates.

$$\mathbb{E}_{p_{\pi^G(\cdot|\cdot,g)(s_T)}}[V^E(s_T)] = \frac{1}{K}\sum_k^K V^E(s_T^k) \qquad \text{where } s_T^k \sim \hat{p}_{\pi^G(\cdot|\cdot,g)(\tau)} \tag{10}$$

$$\hat{p}_{\pi^G(\cdot|\cdot,g)(\tau)} = p(s_0)[\prod_{t=1}^T \hat{M}(s_t|s_{t-1},a_{t-1})\pi^G(a_{t-1}|s_{t-1},g)] \tag{11}$$

The goals sampled for evaluating this exploration potential metric in PEG are drawn from a distribution updated by the MPPI method (Williams et al., 2015; Nagabandi et al., 2020). PEG-G not only uses goals obtained by optimizing Equation 10 to guide exploration sampling but also directly samples trajectories using a goal-conditioned policy with goals provided by the environment. The sampling alternate between these two strategies as shown in the pseudocode in Algorithm 6.

Algorithm 6 PEG-G Sampling

1: **function** COLLECT TRAJECTORIES(...)
2:     **if** episode $i\%2 = 0$ **then**
3:         $g \leftarrow$ Optimize Equation 10 with MPPI
4:         $\tau \leftarrow$ *GO-EXPLORE*(g, π^G, π^E)
5:     **else**
6:         $g \leftarrow$ Returned by environment
7:         $\tau \leftarrow$ Sample a trajectories by π^G using goal g
8: **return** τ

D.4 MEGA-G

MEGA (Pitis et al., 2020) employs kernel density estimates (KDE) to assess state densities and selects goals with low densities from the replay buffer. For the implementation of MEGA, we adopt the model-based MEGA methodology described in the PEG paper without modifications. The PEG paper has illustrated that their adaptation of MEGA outperforms the original MEGA implementation. This entails integrating MEGA's KDE model and incorporating a goal-conditioned value function into the LEXA framework to filter goals based on reachability. Similar to PEG-G, MEGA-G switches between utilizing goals from the environment and employing the MEGA goal selection strategy.

Algorithm 7 MEGA-G Goal Sampling

1: **function** COLLECT TRAJECTORIES(...)
2:     **if** episode $i\%2 = 0$ **then**
3:         $g \leftarrow \min_{g \in \mathcal{D}} \hat{p}(g)$
4:         $\tau \leftarrow$ *GO-EXPLORE*(g, π^G, π^E)
5:     **else**
6:         $g \leftarrow$ Returned by environment
7:         $\tau \leftarrow$ Sample a trajectories by π^G using goal g
8: **return** τ

D.5 MUN-noDAD

We consider the Time-sample Hindsight Waypoints Sampling Strategy from **Hindsight Planner** (Lai et al., 2020) as an alternative to the subgoal selection mechanism in MUN. MUN-noDAD selects

18

subgoals at fixed time intervals along trajectories, providing a simple and effective strategy for defining subgoals. MUN-noDAD can still benefit from the framework of MUN in navigating between different subgoals. The pseudocode for this baseline is as follows:

Algorithm 8 MUN-noDAD Subgoal Picking Strategy

1: **Function:** Subgoals_fixed_interval(...)
2: **Input:** A batch of episodes B_{egc}, number of subgoals $N_{subgoals}$
3: $S_{subgoals} \leftarrow$ pick $N_{subgoals}$ states at fixed time intervals from B_{egc}
4: $G_{subgoals} \leftarrow \eta(S_{subgoals})$
5: **return** $G_{subgoals}$

Algorithm 9 Trainning Frame for MUN-noDAD

1: **Input:** Policy π^G, World Model $\hat{M}$, reward function r^G, subgoals transfer number N_s, subgoal time limit T_s
2: Initialize buffers D, D_{DAD}, D_{egc}
3: **for** $i = 1$ to N_{train} **do**
4:     **if** Should Plan Subgoals **then**
5:         $B_{egc} \leftarrow$ A batch of episodes from D_{egc}
6:         $G_{subgoals} \leftarrow$ Subgoals_fixed_interval(...) with Algorithm 8
7:     Initialize empty trajectory τ
8:     **for** $s = 1$ to N_s **do**
9:         $t_s = 0$
10:         $g_s =$ Sample a subgoal randomly from $G_{subgoals}$
11:         **while** agent has not reached g_s and $t_s < T_s$ **do**
12:             Append one step in real environment with π^G using goal g_s to τ
13:             $t_s \leftarrow t_s + 1$
14:     $D_{DAD} \leftarrow D_{DAD} \cup \{\tau\}$
15:     $D_{egc} \leftarrow D_{egc} \cup$ Sample a trajectory with π^G using goal from training environment
16:     $D \leftarrow D_{DAD} \cup D_{egc}$
17:     Update $\hat{M}$ with D
18:     Update π^G in imagination with $\hat{M}$ to maximize r^G

E   Implementation Details

E.1   Farthest Point Sampling (FPS) Algorithm

Algorithm 10 Farthest Point Sampling (FPS)

1: **function** FPS(points, num_samples)
2:     sampled_points $\leftarrow$ [ ]
3:     first_point $\leftarrow$ random.choice(points)
4:     sampled_points.append(first_point)
5:     min_distances $\leftarrow$ [float('inf')] $\times$ len(points)
6:     **for** each point p in points **do**
7:         min_distances[p] $\leftarrow$ distance(p, first_point)
8:     **for** iteration $i = 1$ to num_samples-1 **do**
9:         farthest_point_index $\leftarrow$ argmax(min_distances)
10:         farthest_point $\leftarrow$ points[farthest_point_index]
11:         sampled_points.append(farthest_point)
12:         **for** each point p in points **do**
13:             min_distances[p] $\leftarrow$ min(min_distances[p], distance(p, farthest_point))
14:     **return** sampled_points

19

The pseudocode presented in the Algorithm 10 illustrate the process of Farthest Point Selection (FPS) algorithm. The FPS algorithm begins by initializing an empty list called 'sampled_points' to store the selected points. The process commences by randomly selecting an initial point from the input point set, denoted as 'points', and adding it to 'sampled_points'. Subsequently, 'min_distances' is initialized to keep track of the minimum distance from each point to any of the sampled points, with initial values set to infinity.

The core procedure involves iteratively selecting points until the desired number of samples is reached. At each iteration, the algorithm identifies the point in 'points' with the maximum minimum distance to the previously sampled points and includes it in 'sampled_points'. Concurrently, 'min_distances' is updated to reflect the recalculated minimum distance of each point to any of the sampled points.

E.2 Runtime

Table 1: Runtimes per experiment.

	Total Runtime (Hours)	Total Steps
3-Block Stacking	70	2.5e6
Walker	40	1.5e6
Ant Maze	36	1e6
Block Rotation	68	2.5e6
Pen Rotation	68	2.5e6
Fetch Slide	52	2e6

We conduct each experiment on GPU Nvidia A100 and require about 3GB of GPU memory. See table in Table 1 for specific running time of MUN for different task. Most of the runtime is consumed by the neural network updates for the policy and the world model, while the time taken by DAD to filter subgoals is minimal.

E.3 Hyperparameters

We use the default hyperparameters of the LEXA backbone MBRL agent (e.g., learning rate, optimizer, network architecture) and keep them consistent across all baselines. MUN primarily requires hyperparameter tuning in the following: 1) the number of candidate subgoals stored $N_{subgoals}$; 2) the number of subgoals used for navigation when sampling in the environment N_s; and 3) the total episode length L and the maximum number of timesteps allocated for navigating to a specific subgoal T_s. We show these hyperparameters in Table 2.

Table 2: Hyperparameters of MUN.

	$N_{subgoals}$	N_s	L	T_s
3-Block Stacking	20	2	150	75
Walker	10	2	150	75
Ant Maze	20	2	500	250
Block Rotation	20	2	150	75
Pen Rotation	20	2	150	75
Fetch Slide	20	2	150	75

F Additional Experiments

F.1 More subgoals found by DAD

We visualize several subgoals found by the DAD algorithm during the training process in Fig. 9. In **Walker**, the first five images show that DAD successfully identifies the crucial joint angles and forces of the Walker robot during its forward locomotion, including standing, striding, jumping, landing,

20

Figure 9: More subgoals found by DAD(Algorithm 2) in all six environments

and leg support. In the subsequent three images, DAD similarly succeeds in recognizing the key movements of the Walker robot during its backward locomotion. In **Ant-Maze**, DAD recognizes significant motion variations at corridor corners. In **Block Rotation** and **Pen Rotation**, DAD is able to identify crucial finger movements subgoals for rotating objects. In **3-Block Stacking**, DAD successfully identifies crucial state transitions required during the stacking process. These critical subgoals include block grasping, lifting, horizontal movement, vertical movement, and gripper release.

F.2 Navigation Experiments

We do the extend navigation experiments on 3-Block Stacking, Ant Maze, and Walker environments to see if the MUN can learn a better world model to navigate to unconstrained goals from unconstrained start state compared to other baselines. In the 3-Block Stacking task, we use a set of 15 goals that represent various critical states in the block-stacking process. These goals serve as candidates for both initial states and endpoint goals, resulting in a total of 225 unique combinations of initial states and endpoint goals for each evaluation episode. For each combination, we conduct 10 repeated evaluations, ultimately computing the average success rate across 2250 evaluation trajectories. Our goal is to assess whether MUN can effectively achieve a random goal when the agent starts from an arbitrary state. This evaluation inherently includes both the forward and reverse processes of stacking blocks, determining whether an agent that can stack blocks is also capable of returning the stacked blocks to an intermediate state. In the Ant Maze environment, we use 32 different positions within the maze as a candidate set for starting and goal positions. Evaluating navigation between these positions allows for a comprehensive assessment of the Ant Robot's world model learning for the maze structure. This evaluation not only measures its ability to reach the final room but also its capability to return to previous rooms from intermediate positions. We evaluate 1024 combinations of starting and goal positions, conducting 10 evaluations for each combination, resulting in an average success rate computed over 10,240 experiments. In the Walker environment, we use all evaluation goals ($\pm13, \pm16, \pm19, \pm22, \pm25, \pm28$) as a candidate set for starting and goal positions. This set can form a total of 144 different combinations of starting and goal positions, providing a thorough assessment of the Walker robot's ability to move forward and backward, as well as its precision in position judgment. See Table 3, 4, 5 for specific results of MUN and all baselines.

21

Table 3: Success rate of navigation experiments on 3-Block Stacking

	Environment	Success rate
MUN	3-Block Stacking	95%
MUN-noDAD	3-Block Stacking	81%
GC-Dreamer	3-Block Stacking	56%
MEGA-G	3-Block Stacking	42%
PEG-G	3-Block Stacking	47%

Table 4: Success rate of navigation experiments on Ant Maze

	Environment	Success rate
MUN	Ant-Maze	96%
MUN-noDAD	Ant-Maze	89%
GC-Dreamer	Ant-Maze	75%
MEGA-G	Ant-Maze	94%
PEG-G	Ant-Maze	93%

Table 5: Success rate of navigation experiments on Walker

	Environment	Success rate
MUN	Walker	89%
MUN-noDAD	Walker	73%
GC-Dreamer	Walker	67%
MEGA-G	Walker	81%
PEG-G	Walker	62%

We observe that MUN significantly outperforms other baselines in navigation experiments across all three environments, demonstrating its exceptional contribution to learning comprehensive world models and policies.

F.3 World Model Assessment

Table 6 shows the single-step prediction error of learned world models. We randomly sample $1e4$ state transition tuples (s_i, a_i, s_{i+1}) within the replay buffers from all of our baselines (MUN, MUN-noDAD, GC-Dreamer, MEGA-G, and PEG-G) to form a validation dataset. Table 6 reports the mean squared error on this dataset.

Table 7 shows the compounding error (multistep prediction error) of learned world models for evaluation when generating the same length simulated trajectories. More specifically, assume a real trajectory of length h is denoted as $(s_0, a_0, s_1, ..., s_h)$. For a learned model M, we sample from s_0 and generate forward rollouts $(\hat{s}_0, a_0, \hat{s}_1, ..., \hat{s}_h)$ where $\hat{s}_0 = s_0$ and for $i \leq 0$, $\hat{s}_{i+1} = M(\hat{s}_0, a_i)$. Then the corresponding compounding error of M is defined as $\frac{1}{h} \sum_{i=1}^{h} \|\hat{s}_i - s_i\|_2^2$. We set h to be the maximum number of timesteps in our environments. We evaluated the compounding prediction error of the learned world models by generating 500 trajectories for each benchmark, simulated on both the models and the real environments.

In Tables 6 and 7, we used the final world models trained by all methods after the same number of environment interaction steps. These results provide a quantitative comparison of the world model prediction quality between MUN and the baselines across our benchmarks. The world models trained by MUN show a much smaller generalization gap to the real environment compared to goal-conditioned Dreamer (and the other baselines). Consequently, MUN can effectively leverage these world models to train control policies that generalize well to the real environment. This explains

22

the superior task success rates of MUN compared to the baselines in our experiment. Fig 10 also provides more information about the world model compound prediction error.

(a) Trajectories 3-Block Stacking   (b) Trajectories Block Rotation
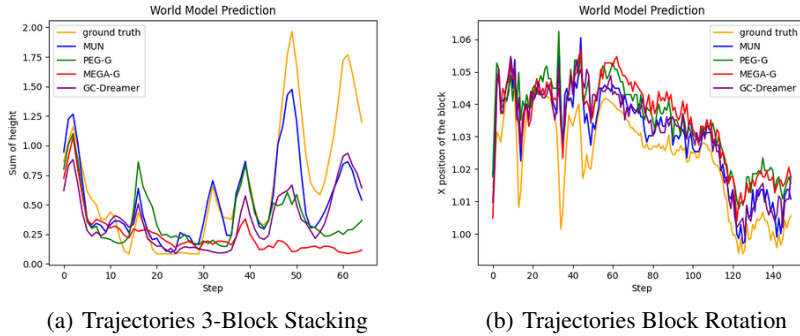
Figure 10: Fig(a) and Fig(b) illustrate the imagined and real environment trajectories for 3-Block Stacking and Block Rotation respectively, starting from the same initial state. Among the baselines, MUN demonstrates the smallest compound model error with respect to the ground truth trajectories. The X-axis represents the trajectory steps. In Fig(a), the Y-axis represents the sum of the heights of the three blocks. MUN's world model outperforms other methods in predicting the correct locations of the three blocks. In Fig(b), the Y-axis represents the position of the block in the x coordinate. MUN's world model outperforms other methods in predicting the correct position of the block.

Table 6: One-step Model Prediction Error.

	MUN	MUN-noDAD	PEG-G	MEGA-G	GC-Dreamer
Ant Maze	1.6740	1.9751	2.1154	2.2416	2.9666
Walker	0.8165	0.9971	1.4759	1.2353	2.1824
3-Block Stacking	0.0070	0.0071	0.0476	0.0853	0.0392
Rotate Block	1.0570	1.5609	1.7753	1.9433	2.3723
Rotate Pen	0.6708	1.1999	1.9622	2.8598	1.8359
Fetch Slide	0.0094	0.0108	0.0132	0.0164	0.0169

Table 7: Compound Model Prediction Error.

	MUN	MUN-noDAD	PEG-G	MEGA-G	GC-Dreamer
Ant Maze	18.83	22.42	29.42	23.69	40.36
Walker	13.03	16.72	26.54	21.21	39.72
3-Block Stacking	0.45	0.55	0.70	0.95	0.94
Rotate Block	11.55	12.86	14.38	14.13	15.06
Rotate Pen	4.63	6.10	7.40	9.85	9.36
Fetch Slide	1.687	1.648	2.195	2.856	2.304

23

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: Our abstract and introduction accurately represent the primary contributions of the paper, which include the development of the MUN algorithm designed to improve goal-conditioned reinforcement learning through enhanced world modeling and exploration capabilities. The introduction outlines the key challenges in GCRL, specifically with sparse rewards, and how MUN addresses these by facilitating effective state transitions between arbitrary subgoal states in the replay buffer. These claims are well-supported by the theoretical underpinnings and experimental results presented in the paper, reflecting the scope and impact of the proposed method.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Our paper thoroughly discusses the limitations of the MUN framework in Appendix. We highlight the dependency on an efficient strategy for discovering key states, pointing out that while DAD is effective, it will also identify irrelevant states in tasks with complex action spaces or weak correlations between goal space and action space. Additionally, we mention the potential for applying MUN to model-free reinforcement learning methods, which do not require learning a world model and have simpler architectures.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.

24

- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [NA]

   Justification: Our paper does not include theoretical results.

   Guidelines: In the Experiment section and the Appendix of our paper, we provide a detailed description of our experimental procedures and configurations. This includes all sources and modifications of the test environments, pseudocode and implementation methods for all baselines, the equipment and memory used, as well as the specific values of the required hyperparameters. Additionally, we have open-sourced our code, which can be found in the Reproducibility Statement section.

   - The answer NA means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: In the Experiment section and appendix of our paper, we elaborate on the procedure and configuration of our experiments. This includes the sources and modifications of all testing environments, pseudo code and implementation methods for all baselines, the devices and memory utilized, as well as specific values of hyperparameters employed. Concurrently, we have open-sourced our code; please refer to the Reproducibility Statement section for further details.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.

25

- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: As we mentioned in the previous justification, we have not only open-sourced our code but also provided detailed steps and settings for reproducing our main experimental results. In the Experiment section and Appendix, we elaborate on the sources and modifications of the environments, baseline implementation details, and MUN implementation specifics.

   Guidelines:
   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: We provide comprehensive details regarding the hyperparameters essential for understanding the experiments, including those specific to our MUN framework. The table presented (Fig 2) outlines these hyperparameters for each task, facilitating reproducibility and comparison.

26

Guidelines:

- The answer NA means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: We conducted each experiment a minimum of five times using different random seeds, and upon plotting the results, as demonstrated in the Experiment section, we incorporated the experimental error. The solid line denotes the average success rate, while the shaded region signifies the standard deviation among the repeated experimental outcomes.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [Yes]

Justification: We clearly specifies the computer resources (Nvidia A100 GPU) and the amount of GPU memory required (approximately 3GB). Additionally, we provides detailed information on the runtime of each experiment in Appendix.

Guidelines:

- The answer NA means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

27

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research conducted in our paper aligns with the NeurIPS Code of Ethics. We have thoroughly reviewed the guidelines and ensured that our research adheres to ethical standards. Additionally, we have implemented measures to safeguard anonymity and comply with pertinent laws and regulations.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [NA]

    Justification: Our research aims to address the exploration problem in Reinforcement Learning (RL) within the GCRL environment. It is currently in the theoretical research stage and has minimal societal impact.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: Our paper poses no such risks.

    Guidelines:

    - The answer NA means that the paper poses no such risks.

28

- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: Our paper properly credits the creators or original owners of assets used, including code, data, and models. The licenses and terms of use are explicitly respected. Specifically, we cite the original papers for code packages or datasets used, state the version of the assets, and include URLs where possible.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: We have documented our code and provided detailed instructions on its usage, licenses, and permissible scope of use. Additionally, we have included the documentation alongside the assets to ensure accessibility and clarity for users.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

29

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [NA]

Justification: Our paper not involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: Our paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

30