Exploring the Edges of Latent State Clusters for Goal-Conditioned Reinforcement Learning

Yuanlin Duan
Rutgers University
yuanlin.duan@rutgers.edu

He Zhu
Rutgers University
hz375@cs.rutgers.edu

Abstract

Exploring unknown environments efficiently is a fundamental challenge in unsupervised goal-conditioned reinforcement learning. While selecting exploratory goals at the frontier of previously explored states is an effective strategy, the policy during training may still have limited capability of reaching rare goals on the frontier, resulting in reduced exploratory behavior. We propose "Cluster Edge Exploration" (CE^2), a new goal-directed exploration algorithm that when choosing goals in sparsely explored areas of the state space gives priority to goal states that remain accessible to the agent. The key idea is clustering to group states that are easily reachable from one another by the current policy under training in a latent space, and traversing to states holding significant exploration potential on the boundary of these clusters before doing exploratory behavior. In challenging robotics environments including navigating a maze with a multi-legged ant robot, manipulating objects with a robot arm on a cluttered tabletop, and rotating objects in the palm of an anthropomorphic robotic hand, CE^2 demonstrates superior efficiency in exploration compared to baseline methods and ablations.

1 Introduction

In recent years, Goal-Conditioned Reinforcement Learning (GCRL) (Andrychowicz et al. (2017)) has emerged as a powerful paradigm for training agents to accomplish diverse tasks in complex and dynamic environments. GCRL enables agents to learn goal-directed behaviors, allowing them to achieve specific objectives in a flexible and adaptive manner. However, a central challenge in GCRL lies in guiding agents to effectively explore their environment during training. The exploration problem in GCRL can be viewed as the task of setting goals for the agent during training to guide the agent's environment navigation to collect exploratory data that improves its learning process. In this paper, we address this critical challenge by proposing a novel strategy for selecting exploration-inducing goals in GCRL.

Because goal-conditioned policies excel at reaching states encountered frequently during training, a simple strategy is setting goals in less-visited areas of the state space to broaden the range of reachable states. However, throughout training, goal-conditioned policies may encounter difficulties in reaching arbitrary goals. For example, when instructed to navigate to an unexplored section of a maze environment, a novice agent might instead revisit a previously traversed area that provides low exploration value. To address this shortcoming, the environment exploration procedure must set up additional mechanisms to filter out unreachable goals. A common strategy in the literature is to select goals at the frontier of previously explored states and launch an exploration phase immediately after these goals are achieved, adhering to a Go-Explore principle (Ecoffet et al. (2019)). For example, Skewfit (Pong et al. (2019)) estimates state densities and selects goals at the frontier from the replay buffer in inverse proportion to their density. Similarly, MEGA (Pitis et al. (2020)) uses kernel density estimates (KDE) of state densities and selects frontier goals with low density from the replay buffer.

Preprint. Under review.

However, precisely identifying the frontier of known states can be challenging with these heuristics. Even once the frontier is identified, the policy during training may still have limited capability of reaching rare goals on the frontier, resulting in reduced exploratory behavior.

To address the aforementioned challenge, we propose a new goal-directed exploration algorithm, CE^2 (short for "Cluster Edge Exploration"). When choosing goals in sparsely explored areas of the state space, CE^2 gives priority to goal states that remain accessible to the agent. For this purpose, our key idea is clustering to group known states that are easily reachable from one another by the current policy under training, and traversing to states holding significant exploration potential on the boundary of these clusters before doing exploratory behavior. In this way, our method accounts for the capability of the current policy for exploratory goals. First, a state cluster likely represents part of the state space where the training policy is familiar with. Second, given the easy accessibility of states within each cluster by the training policy, the agent's capability extends to reaching states even at cluster boundaries. Moreover, less explored regions naturally reside adjacent to the periphery of state clusters. This Go-Explore strategy enables the agent to progressively broaden the coverage of each state cluster to effectively explore a novel environment. We instantiate CE^2 in the context of model-based GCRL, demonstrating how learned world models can facilitate clustering environment states that are easily reachable from one another by the training policy in a latent space. We validate the effectiveness of CE^2 in challenging robotics scenarios, including navigating a maze with a multi-legged ant robot, manipulating objects with a robot arm on a cluttered tabletop, and rotating objects in the palm of an anthropomorphic robotic hand. In each scenario, CE^2 exploration results in more efficient training of adaptable GCRL policies compared to baseline methods and ablations.

2 Problem Setup and Background

Our work focuses on the exploration problem in unsupervised goal-conditioned reinforcement learning (GCRL) settings. In this section, we set up notation and preliminary concepts.

GCRL. A goal-conditioned Markov decision process (MDP) is defined by the tuple (S, A, G, T, η) where the state space S defines the set of all possible agent's observations into the environment, the action space A defines all possible actions that the agent can take in each state, G is the set of all possible goals that the agent may aim to achieve in the environment, and the transition function T describes the probability of transitioning from one state to another given an action. It is defined as $T(s'|s, a)$, where $s' \in S$ is the next state, $s \in S$ is the current state, and $a \in A$ is the action taken. $\eta : S \to G$ is a tractable mapping function that maps a state to a specific goal. A goal-conditioned $\pi(a|s, g)$ represents the agent's strategy for selecting actions based on states and goal commands, indicating the probability of taking action a in state s given goal command $g \in G$.

In this paper, for ease of presentation, we assume $S = G$ and η is an identify function.

Our goal is to develop agents capable of unsupervised exploration when dropped into an unknown environment. During the unsupervised exploration stage, there are no predefined tasks or goals, the agent sets its own goal command $g \in G$ as it explores the environment. Following this exploration phase, a successful agent should be able to navigate to a wide range of previously unknown goal states in the environment upon goal commands.

Model-based GCRL. Model-based reinforcement learning (MBRL) is an approach where an agent learns a model of the environment's dynamics to predict future states, enabling more efficient policy learning. Fig. 1 shows 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 B.1. Particularly, we consider **GC-Dreamer** (goal-conditioned Dreamer). In GC-Dreamer, the goal-conditioned agent $\pi^G(a|s, g)$ samples goal commands $g \in G$ from a given environment goal distribution p_g to collect trajectories in the real world. These trajectories are used to

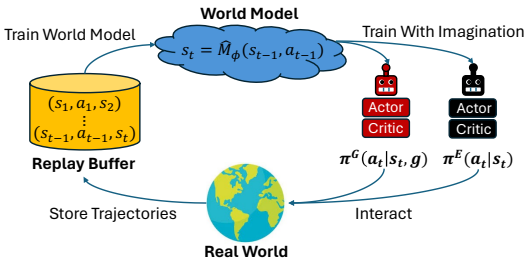


Figure 1: Model-based GCRL Framework

train the world model $\hat{M}$, and subsequently, π^G is trained on imagined rollouts generated by $\hat{M}$, with these two steps run in alternation. The reward function used to train π^G is determined by a temporal distance network D_t (see below).

Go-Explore. In unsupervised GCRL, the goal distribution p_g is only revealed at test time. "Go-Explore" (Ecoffet et al. (2019); Pislar et al. (2022); Tuyls et al. (2022); Hu et al. (2023)) is a popular mechanism tailored for long-term GCRL scenarios that require intensive exploration. The Go-Explore methodology splits each training episode into two distinct phases: the "Go-phase" and the "Explore-phase". In the "Go-phase", the agent is guided to an "interesting" goal g (Pong et al. (2019); Pitis et al. (2020)) (e.g., states rarely encountered in the replay buffer) by the GCRL policy π^G, reaching a final state s_T. Subsequently, the "Explore-phase" kicks in, with an undirected exploration policy π^E taking over from s_T for the remaining timesteps. This exploration policy is optimized to maximize an intrinsic exploration reward (Bellemare et al. (2016); Pathak et al. (2017); Burda et al. (2018); Sekar et al. (2020)) (e.g., to explore less familiar areas of the environment that the world models haven't adequately learned).

Recently, Go-Explore has been integrated with model-based unsupervised GCRL (Mendonca et al. (2021); Hu et al. (2023)), as depicted in Fig. 1. In addition to the goal-conditioned policy $\pi^G(a|s,g)$, an exploration policy $\pi^E(s)$ is introduced into the model-based GCRL framework. The agent's training process involves learning the following components:

World Model:	$\hat{M}(s_t\|s_{t-1}, a_{t-1})$		
Exploration policy:	$\pi^E(s_t)$	Goal Reaching policy:	$\pi^G(s_t, g)$
Exploration value:	$V^E(s_t)$	Goal Reaching value:	$V^G(s_t, g)$

where both π^G and π^E are trained using the model-based actor-critic algorithm in Dreamer (Hafner et al. (2020)). They are entirely trained with the imagined rollouts of the world model $\hat{M}$ to maximize the accumulated rewards $\sum_t r_t^G$ and $\sum_t r_t^E$, respectively. The explorer reward r^E encourages exploration by leveraging the Plan2Explore (Sekar et al. (2020)) disagreement objective, which motivates the agent to seek states that induce discrepancies among an ensemble of world models. In contrast, the goal-reaching reward r^G is driven by the self-supervised temporal distance objective D_t (Mendonca et al. (2021)), which reinforces the policy to minimize the action steps required to transition from the current state s to a sampled goal state g in an imagined rollout, i.e. $r^G(s,g) = -D_t(\Psi(s), \Psi(g))$. The temporal distance network D_t predicts the anticipated number of action steps needed to transition from s to g. It is trained by extracting pairs of states s_t and s_{t+k} from an imagined rollout generated by $\hat{M}$ and predicting the distance k as follows:

$$D_t\big(\Psi(s_t), \Psi(s_{t+k})\big) \approx k \tag{1}$$

Here, Ψ is a learned function for state embeddings in the world model (We assume $S = G$ in the paper). Further details on the training procedure of D_t can be found in Appendix B.2.

CE^2 aims to address the core challenge in the Go-Explore mechanism: how do we select an interesting goal command g at the frontier of known states with high exploration potential and effectively guide the agent to g?

3 State Cluster Edge Exploration

The major limitation in existing Go-Explore approaches, such as those described in (Pong et al. (2019); Pitis et al. (2020)) is that the policy under training can struggle to reach heuristically chosen rare goals at the frontier of known states (Hu et al. (2023)). This difficulty arises because the goal commands are selected without a systematic method to filter out unachievable goals for the agent, leading to diminished exploratory behavior. In CE^2 , when choosing goals in sparsely explored areas of the state space in the "Go-phase", our method gives priority to goal states that remain accessible. For this purpose, the key idea is clustering to group states that are easily reachable from one another by the current policy under training in a latent space, and selecting states holding significant exploration potential on the boundary of these clusters as the "interesting" goals to explore. In Sec. 3.1, we discuss how to learn a latent space that can represent the reachability relationships between environment states. In Sec. 3.2, we explain how this latent space can be used to cluster states in the replay buffer that are easily reachable from one another. In Sec. 3.3, we demonstrate how the agent can be brought to interesting states on the boundary of latent state clusters to effectively explore its environment.

3.1 Latent Space Learning

Typically, during the learning process of a world model $\hat{M}$ as a neural network, an essential step involves encoding states from the original observation space into a latent space using an encoder, which can then be decoded back to the original observation space by a decoder. This latent space is subsequently used to learn the dynamic model of the real environment Hafner et al. (2019a).

In CE^2 , we additionally require the latent space can express the temporal distance between different states. In other words, we aim for the distances between various states in the latent space to represent the number of steps required to transition from one another in the real environment (after decoding) by the training policy. Therefore, the loss function of training the latent space in CE^2 comprises two components. The first component is the reconstruction loss $\mathcal{L}_{rec}$, akin to the latent space loss function in Dreamer framework (Hafner et al. (2019a)). It captures the association between the latent space and the re-decoding to the observation space, along with predicting dynamic transition in the latent space. We introduce a second loss term $\mathcal{L}_{dt}$ that leverage the temporal distance network D_t in Equation 1 to guide the learning of the latent space structure. For any pair of states $(s1, s2)$ sampled from the replay buffer, the $\mathcal{L}_{dt}$ loss function is formulated as follows (Ψ is a learned function for state embeddings in the world model):

$$\mathcal{L}_{dt} = (\|\Psi(s_1) - \Psi(s_2)\|_2^2 - \frac{1}{2}(D_t(\Psi(s_1), \Psi(s_2)) + D_t(\Psi(s_2), \Psi(s_1))))^2 \tag{2}$$

$$\mathcal{L}_{latent} = \mathcal{L}_{rec} + \mathcal{L}_{dt} \tag{3}$$

We use the loss function $\mathcal{L}_{latent}$ to supervise the training of the latent space. The trained latent space provides the agent with a deeper understanding of the real environment, where states that are easily reachable from one another in the real environment are closer in proximity within the latent space.

3.2 Latent State Clustering

To identify the frontier of known states, CE^2 conducts state clustering to group states in the replay buffer. States that are easily reachable from one another are classified in the same cluster in the latent space by Gaussian Mixture Models (GMMs), based on the temporal distances between the encoded states. GMMs are probabilistic models that assume all data points are generated by a mixture of a finite number of Gaussian Distributions. We initialize the Gaussian models in the latent space with N_c trainable latent centroids $c = \{c_1, \ldots, c_{N_c}\}$ and a shared variance σ, where N_c represents the desired number of clusters. These N_c latent centroids are initialized by applying the Farthest Point Sampling (FPS) algorithm(Eldar et al. (1997)) to select a representative subset of states from a batch of data sampled from the replay buffer. We provide a detailed description of the FPS algorithm in the Appendix F.1. After initialization, we optimize the clustering model by maximizing the Evidence Lower Bound (ELBO) iteratively on sampled batches from the replay buffer with a uniform prior $p(c)$ to scatter out the latent centroids (Zhang et al. (2021)):

$$\log p(z = \Psi(s)) \geq \mathbb{E}_{q(c|\Psi(s))}[\log p(\Psi(s)|c)] - D_{KL}(q(c|\Psi(s))||p(c)) \tag{4}$$

3.3 Exploring the Boundaries of Latent State Clusters

Assuming we have already trained N_c state clusters in the latent space, each representing part of the state space where the goal-conditioned policy under training is familiar with, how can we utilize these state clusters to plan an exploration strategy? CE^2 selects goal states at the edges of these latent state clusters for exploration because (1) less explored regions are naturally adjacent to these boundaries, and (2) given the easy accessibility between states within each cluster by the training policy, the agent's capability extends to reaching states even at the cluster boundaries.

We outline our exploration algorithm in Algorithm 1. At line 3, it samples $N_{candidate}$ latent states as $S_{candidate}$ from GMMs. A higher sampling quantity ensures sampling from more states at the edges of the clusters. We set $N_{candidate} = 1000$ in CE^2 . We compute the total probability of each latent state $\hat{s} \in S_{candidate}$ in the Gaussian mixture model, given by the formula:

$$p(\hat{s}) = \sum_{i=1}^{N_c} \beta_i \mathcal{N}(\hat{s}|c_i, \sigma) \tag{5}$$

In this formula, β_i are the mixture weights satisfying $\beta_i \geq 0$ and $\sum_{i=1}^{N_c} \beta_i = 1$, $\mathcal{N}(\hat{s}|c_i, \sigma)$ represents the i-th Gaussian distribution with mean c_i and the shared standard deviation σ. At line 4, we select N_{edge} latent states with the lowest total probability from $S_{candidate}$ by Equation 5 as a set S_{edge}. Intuitively, these states reside on the edges of the latent state clusters and, therefore, induce a set of a goal commands $G_{edge} = \{\eta(f_D(\hat{s}))|\hat{s} \in S_{edge}\}$ that may be used for the "Go-phase" for Go-Explore, where f_D is the state decoder and η is the goal mapping function. However, randomly picking a goal command from G_{edge} overlooks whether the policy can exactly navigate the agent to the sampled goal in the real environment. Although determining the exact outcome of the policy without execution is impractical, similar to PEG (Hu et al. (2023)), we can leverage the world model to provide an approximation of the exploration potential $P^E(g)$ of a goal command g:

Algorithm 1 Cluster Edge Exploration(CE^2)

1: $D_{exp} \leftarrow \{\}$
2: **for** episode $i = 1$ to N_τ **do**
3: $S_{candidate} \leftarrow$ Sample $N_{candidate}$ points from GMM
4: $G_{edge} \leftarrow N_{edge}$ states in $S_{candidate}$ with the smallest total probability based on Equation 5.
5: $g^E \leftarrow argmax_{g \in G_{edge}} P^E(g)$ through imagination with $\hat{M}$
6: $\tau \leftarrow$ *GO-EXPLORE*(g^E, π^G, π^E)
7: $D_{exp} \leftarrow D_{exp} \cup \tau$

$$\hat{p}_{\pi^G(\cdot|\cdot,g)(\tau)} = p(s_0)[\prod_{t=1}^{T} \hat{M}(s_t|s_{t-1}, a_{t-1})\pi^G(a_{t-1}|s_{t-1}, g)] \tag{6}$$

$$P^E(g) = \mathbb{E}_{p_{\pi^G(\cdot|\cdot,g)(s_T)}}[V^E(s_T)] \quad \approx \frac{1}{K}\sum_{k}^{K} V^E(s_T^k) \qquad \text{where } s_T^k \sim \hat{p}_{\pi^G(\cdot|\cdot,g)(\tau)} \tag{7}$$

In Equation 6, we simulate the "Go-phase" of Go-Explore over the world model $\hat{M}$. We set each state from G_{edge} as the goal command g for the goal-conditioned policy π^G to run over $\hat{M}$ and denote s_T as the final state of the resulting imagined trajectory (here $\hat{p}_{\pi^G(\cdot|\cdot,g)(\tau)}$ essentially induces the imagined trajectory distribution over the world model). We use the learned exploration value function V^E of explorer π^E to estimate the exploration value of s_T^k, the final state of k-th imagined trajectory. We average the estimated exploration potential over K such imagined trajectories.

At line 5 in Algorithm 1, after selecting the exploration target g^E with the highest exploration potential P^E from the latent cluster boundaries, we start the Go-Explore procedure in the real environment by executing the goal-conditioned policy π^G to approach g^E as closely as possible limited in T timesteps, followed by launching the explore policy π^E for exploration limited in T_E timesteps.

3.4 The Main Algorithm

We depict the main learning algorithm of CE^2 in Algorithm 2. Recall that the learning objective is to train an agent that can achieve diverse goals revealed to it only at test time. Accordingly, in this algorithm at line 7, the data D_{exp} collected to train the world model $\hat{M}$ is generated solely by our Go-Explore strategy as outlined in Algorithm 1. At line 6, we periodically update the centroids of the latent clusters again using FPS algorithm (Eldar et al. (1997)) from a batch of latest trajectories from the replay buffer. This ensures that the candidate goal states selected for exploration are indeed located at the boundaries of key state regions. At line 10, we train the clustering model using data from the replay buffer in each round. This ensures that latent state clustering and the agent's goal-reaching capability are kept synchronized.

Algorithm 2 The main training algorithm for CE^2

1: **Input:** π^G, π^E, World Model $\hat{M}$, GMM, r^G, r^E
2: Initialize replay buffer D
3: **for** $i = 1$ to N_{train} **do**
4: **if** Should assign centroids **then**
5: $B_{exp} \leftarrow$ A batch of data from D_{exp}
6: GMM $\leftarrow$ Choose N_c centroids from B_{exp} by FPS
7: $D_{exp} \leftarrow$ Cluster Edge Exploration(...) with Algorithm 1
8: $D \leftarrow D \cup D_{exp}$
9: Update $\hat{M}$ with D (update latent space by $\mathcal{L}_{rec} + \mathcal{L}_{latent}$)
10: Update GMM with D_{exp}
11: Update π^G in imagination with $\hat{M}$ to maximize r^G
12: Update π^E in imagination with $\hat{M}$ to maximize r^E

In our experiment, we also designed a variant of CE^2 in Algorithm 3, called CE^2-G . This algorithm is given the environment goal distribution p_g at training time. The main idea is to progressively expand the scope of exploration around the possible trajectories leading to the environment goals. In this algorithm, the replay buffer additionally include D_{egc} the trajectories sampled by π_G conditioned on the environment goals in p_g. We only use D_{egc} to initialize and train latent state clusters. In this

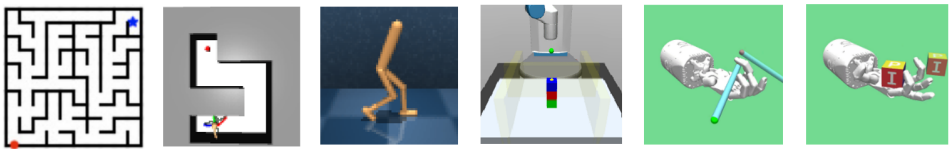


Figure 2: We conduct experiments on 6 environments: Point Maze, Ant Maze, Walker, 3-Block Stacking, Block Rotation, Pen Rotation.

way, the agent is encouraged to prioritize exploration starting from the edges of latent state clusters along the trajectories towards the goal states in p_g. CE^2-G can be considered as learning policies and world models specific to a given goal distribution.

4 Experiments

Our experiments evaluate goal-reaching tasks that demand significant exploration to solve. We aim to address the following questions: (1) Does CE^2 lead to improved exploration and goal-reaching performance? (2) How do CE^2 exploration qualitatively differ from those in previous goal-directed exploration methods? (3) Which components of CE^2 are crucial to its success?

Algorithm 3 The main training algorithm for CE^2-G

Input: π^G, π^E, G, World Model $\hat{M}$, GMM, r^G, r^E
Initialize replay buffer D
for $i = 1$ to N_{train} **do**
  if Should assign centroids **then**
    $B_{egc} \leftarrow$ A batch of data from D_{egc}
    $GMM \leftarrow$ Choose N_c centroids from B_{egc} by FPS
  $D_{exp} \leftarrow$ Cluster Edge Exploration(...) with Algorithm 1
  $D_{egc} \leftarrow$ Rollouts of π^G using the env goal distribution p_g
  $D \leftarrow D \cup D_{exp} \cup D_{egc}$
  Update $\hat{M}$ with D (update latent space by $\mathcal{L}_{rec} + \mathcal{L}_{latent}$)
  Update GMM with D_{egc}
  Update π^G in imagination with $\hat{M}$ to maximize r^G
  Update π^E in imagination with $\hat{M}$ to maximize r^E

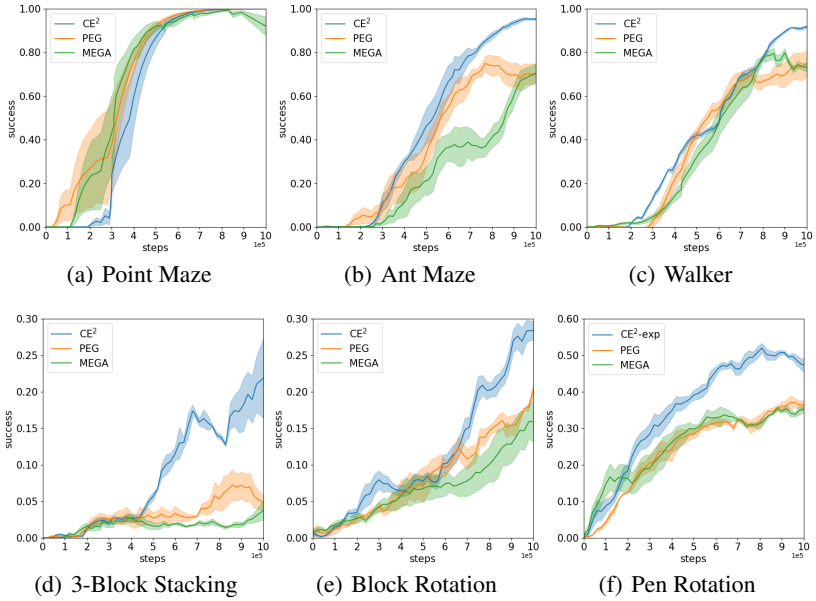
4.1 Benchmarks

We evaluate our method on six hard exploration goal-conditioned RL tasks: **Point-Maze**, **Ant-Maze**, **Walker**, **3-Block Stacking**, **Block Rotation** and **Pen Rotation**. **Point-Maze**: A blue point is placed at the bottom left of the maze and be trained to explore the structure of maze. **Ant-Maze**: An ant robot must master intricate four-legged locomotion behaviors and maneuvering through narrow hallways. **Walker**: A 2-legged robot needs to learn how to control its leg joints to walk on a flat plane to move 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 pushing, picking, and stacking, as well as discovering intricate action paths to accomplish the task within the environment. Previous solutions have relied on methods like demonstrations, curriculum learning, or extensive simulator data, highlighting the task's difficulty. The Gymnasium **Block Rotation** and **Pen Rotation** tasks involve manipulating a block and a pen respectively to achieve a random target rotation along all axes. Pen Rotation is particularly challenging due to the pen's thinness, requiring precise control to prevent it from dropping. For evaluation, we use the most challenging goals, such as the farthest goal locations, in Point Maze, Ant Maze, Walker, and 3-Block Stacking. In the other two environments, we utilize random goals as defined by the environment. For more settings and information about the environments, please refer to the Appendix D.

4.2 Baselines

In the unspervised GCRL setting, we compared **CE^2** with the Go-Explore strategy (Ecoffet et al. (2019)) that has been proved efficient in this setup. We consider the most closely relevant Go-Explore baselines **PEG** (Hu et al. (2023)) and **MEGA** (Pitis et al. (2020))[1]. 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 so that the agent's goal-conditioned policy, given its current level of training, reaches states with high exploration potential. This potential is defined as the expected accumulated exploration reward during the Explore-phase.

[1]Our model-based MEGA baseline is borrowed from (Hu et al. (2023))



(a) Point Maze (b) Ant Maze (c) Walker

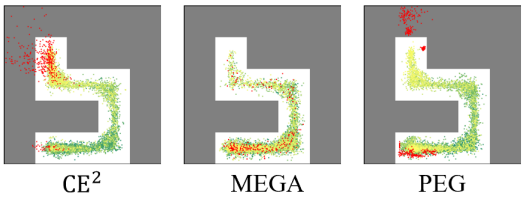
(d) 3-Block Stacking (e) Block Rotation (f) Pen Rotation

Figure 3: Experiment results comparing CE^2 with the baselines over 5 random seeds.

In scenarios where environment goal distributions are available to the agents, we compare CE^2-G with **GC-Dreamer** (illustrated in Sec. 2), **PEG-G**, **MEGA-G** and **L3P**. Similar to CE^2-G , 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 goal-conditioned policy π_G commanded by environment goals but also exploratory trajectories sampled using the corresponding Go-Explore strategies. **L3P** trains a latent space using temporal distances and performs clustering in this latent space, similar to CE^2-G . However, **L3P** does not employ a Go-Explore strategy. Instead, it constructs a directed graph with cluster centroids as nodes and utilizes online planning with graph search to determine subgoals for task execution.

4.3 Results

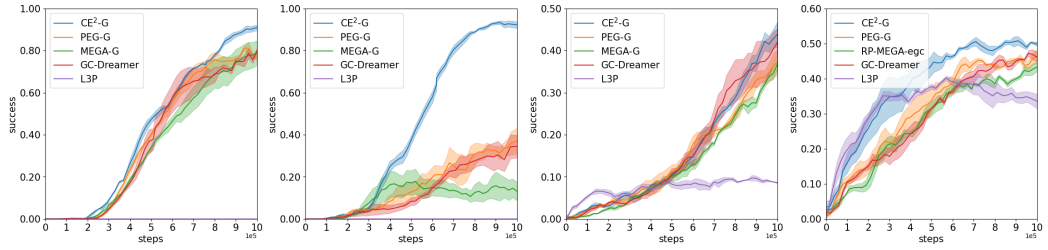
CE^2 Results. Fig. 3 depicts the mean learning performance of all the unsupervised GCRL tools in terms of the agent's goal-reaching success rate averaged over 5 random seeds over 1e6 timesteps. The evaluation goal distribution is revealed to the agent only at test time. In all tasks except PointMaze, CE^2 significantly outperforms PEG and MEGA in terms of both learning performance and learning speed. On PointMaze, CE^2 performs comparably with the baselines. Although MEGA can set goal commands in sparsely explored areas of the state space to encourage exploration, unlike CE^2 , it lacks a systematic method to filter out unachievable goals for the agent, which can result in inefficient exploration. Theoretically, PEG can induce more exploration than MEGA because it can sample goal commands as any states within the state space to initiate exploration, including those beyond the frontier of known states in the replay buffer. However, because a learned world model is typically unfamiliar with rarely observed states, it may select goal commands that appear to have high exploration potential in the model but perform poorly in the real environment as shown in Fig. 4.



CE^2 MEGA PEG

Figure 4: Comparison of exploration goals (represented as red points) generated by CE^2 , MEGA, and PEG in the Ant Maze

CE^2-G Results. Fig. 5 depicts the mean learning performance of all the tools in terms of the agent's goal-reaching success rate averaged over 5 random seeds over 1e6 timesteps, when the environment goal distribution is revealed to the agent at training time. GC-Dreamer is the only tool that lacks a



(a) Walker (b) 3-Block Stacking (c) Block Rotation (d) Pen Rotation

Figure 5: Experiment results comparing CE^2-G with the baselines over 5 random seeds.

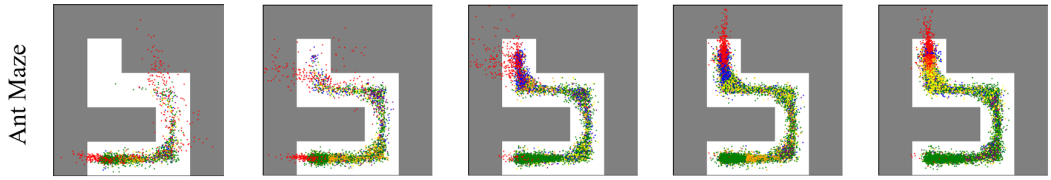


Figure 6: Cluster evolution in CE^2 as the training progresses. The red points means the goals picked by CE^2 to explore and other points in different colors represent the clusters CE^2 learned.

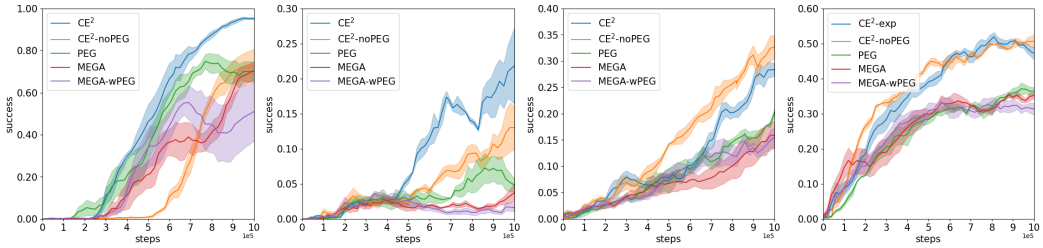
Go-Explore phase, which may limit its exploration potential. Even so, it can sometimes outperform MEGA-G and PEG-G (see block rotation and pen rotation). This indicates that, without reasonably accounting the agent's capability to reach selected goal commands, the Go-Explore strategy does not always guarantee improved exploration. Suboptimal goal-setting during the "Go-phase" can even hinder exploration (see 3 block stacking). Notably, for the challenging 3 block stacking task, CE^2-G achieves a high success rate exceeding 90%. In comparison, MEGA-G, PEG-G and GC-Dreamer only achieves less than 40% success rates. Refer to Appendix for full results of CE^2-G .

4.4 Exploration Process

Figure 6 shows the evolution of state clusters (learned in a latent space) during the training process for Ant Maze (in different colors). The red points represent the selected goal commands used to induce exploration. We observe that the self-directed exploration goals improve progressively as the agent's capabilities increase, consistently targeting the cluster edges that require further exploration and are within the agent's reach. We also compare the exploration targets generated by CE^2 with those produced by the MEGA and PEG approaches throughout the training process in the Ant Maze environment and present more discussion in Appendix G.2.

4.5 Ablation Study

In the ablation experiment, our goal is to determine the individual contributions of each component to our method's overall performance. The "Go-phase" of the Go-Explore procedure in CE^2 consists of two main steps for selecting a goal command g to initiate exploration: (a) sampling environment states at the boundaries of its trained latent state clusters, and (b) selecting the goal command g with the highest exploration potential from the sampled states. Our first ablation, **CE^2-noPEG** , only performs step (a). It randomly samples g from the latent state clusters without considering its exploration potential. The second ablation only performs step (b) and is identical to the PEG baseline. It can sample any state within the state space for the goal command g and is not limited to commanding the agent to novel or rarely observed states at the frontier of known states like CE^2 and MEGA. We also include **MEGA** and **MEGA-wPEG** as two baselines to solely compare the exploration strategy-step (a)-in CE^2 with MEGA's strategy to command the agent to rarely seen states. MEGA-wPEG first uses MEGA to sample a batch of candidate goals, all with low density in the replay buffer. Then, their exploration potential is evaluated using PEG (step (b)), and the most



(a) Ant Maze (b) 3-Block Stacking (c) Block Rotation (d) Pen Rotation

Figure 7: Ablation study on the importance of each component of CE^2 over 5 random seeds.

valuable one is selected as the exploratory goal. We conduct the ablation experiments in a purely unsupervised setting, without revealing any test goals at training time.

Fig. 7 confirms that both step (a) and step (b) in CE^2 are important. CE^2 significantly outperforms CE^2-noPEG and PEG in 3-block stacking and the Ant maze tasks. Notably, even without step (b), CE^2-noPEG performs well across all experiments, especially in the challenging block and pen rotation tasks. This indicates that the goal commands sampled at the edges of latent state clusters already possess high exploration potential and can guide the agent to traverse unseen state spaces. The superior performance of CE^2-noPEG compared to both MEGA and MEGA-wPEG further reinforces this. MEGA achieves similar or better performance compared to MEGA-wPEG, indicating that PEG's effectiveness relies on the quality of the candidate goal set. The exploratory goals sampled from the lowest-density regions in the replay buffer might be beyond the agent's capability, leading PEG to inaccurately assess the true exploration potential of the candidate goals.

We also conducted experiments in the CE^2 with different numbers of latent state clusters N_c and observed that CE^2 is insensitive to this hyperparameter.See Appendix G.5 for more discussion.

5 Related Work

Our method addresses the challenging and inefficient exploration problem inherent in goal-conditioned reinforcement learning (RL) settings with sparse rewards, commonly used in robotics and control fields (Ghosh et al. (2020); Liu et al. (2022); Plappert et al. (2018)). In goal-conditioned RL, agents are trained to achieve various goals based on predefined commands, with rewards typically being binary, indicating positive feedback from the environment only upon reaching the specified goal. This sparse reward setting significantly complicates achieving sample efficiency and effective learning processes (Ren et al. (2019); Florensa et al. (2018); Trott et al. (2019)). To mitigate this challenge, various methods have been proposed. Some reshape the sparse reward function into a denser form by incorporating metrics such as distance between achieved and desired goals(Trott et al. (2019)) or temporal distance (Hartikainen et al. (2019); Mendonca et al. (2021)). Additionally, exploration strategies often include rewards aimed at incentivizing visits to states with low visitation frequencies (Bellemare et al. (2016); Burda et al. (2018)). These approaches typically involve identifying states with infrequent occurrences within the replay buffer and targeting them for exploration, thus facilitating the discovery of unknown regions in the environment. Furthermore, some research emphasizes the exploration of states with high variance between ensemble predictions of future states (McCarthy et al. (2021); Oudeyer et al. (2007); Pathak et al. (2017); Henaff (2019); Shyam et al. (2019); Sekar et al. (2020)). See Appendix A for more related work discussion.

6 Conclusion

We present CE^2 , a novel Go-Explore mechanism designed to tackle hard exploration problems in unsupervised goal-conditioned reinforcement learning tasks. While CE^2 outperforms prior exploration approaches in challenging robotics scenarios, the requirement to learn state clusters to identify frontier states and the reliance on world models to determine exploration potential introduce nontrivial computational costs. Exploring whether CE^2 's Go-Explore strategy can be effectively applied to model-free GCRL settings remains an interesting avenue for future work.

References

Andrychowicz, M., Crow, D., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, O. P., and Zaremba, W. (2017). Hindsight Experience Replay.

Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. (2016). Unifying Count-Based Exploration and Intrinsic Motivation. In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc.

Buckman, J., Hafner, D., Tucker, G., Brevdo, E., and Lee, H. (2019). Sample-Efficient Reinforcement Learning with Stochastic Ensemble Value Expansion.

Burda, Y., Edwards, H., Storkey, A., and Klimov, O. (2018). Exploration by Random Network Distillation.

Chaslot, G. M.-B., Winands, M. H., Szita, I., and Van Den Herik, H. J. (2008). CROSS-ENTROPY FOR MONTE-CARLO TREE SEARCH. *ICGA Journal*, 31(3):145–156.

Chua, K., Calandra, R., McAllister, R., and Levine, S. (2018). Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models.

Cui, G. and Zhu, H. (2021). Differentiable Synthesis of Program Architectures. In *Advances in Neural Information Processing Systems*, volume 34, pages 11123–11135. Curran Associates, Inc.

Deisenroth, M. P. and Rasmussen, C. E. (2011). PILCO: A Model-Based and Data-Efficient Approach to Policy Search.

Ecoffet, A., Huizinga, J., Lehman, J., Stanley, K. O., and Clune, J. (2019). Go-Explore: A New Approach for Hard-Exploration Problems.

Eldar, Y., Lindenbaum, M., Porat, M., and Zeevi, 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.

Ghosh, D., Gupta, A., Reddy, A., Fu, J., Devin, C., Eysenbach, B., and Levine, S. (2020). Learning to Reach Goals via Iterated Supervised Learning.

Gu, J., Xiang, F., Li, X., Ling, Z., Liu, X., Mu, T., Tang, Y., Tao, S., Wei, X., Yao, Y., Yuan, X., Xie, P., Huang, Z., Chen, R., and Su, H. (2023). Maniskill2: A unified benchmark for generalizable manipulation skills.

Guan, L., Valmeekam, K., Sreedharan, S., and Kambhampati, S. (2023). Leveraging Pre-trained Large Language Models to Construct and Utilize World Models for Model-based Task Planning. In *Thirty-Seventh Conference on Neural Information Processing Systems*.

Ha, D. and Schmidhuber, J. (2018). Recurrent World Models Facilitate Policy Evolution.

Hafner, D., Lillicrap, T., Ba, J., and Norouzi, M. (2019a). Dream to Control: Learning Behaviors by Latent Imagination.

Hafner, D., Lillicrap, T., Fischer, I., Villegas, R., Ha, D., Lee, H., and Davidson, J. (2019b). Learning Latent Dynamics for Planning from Pixels.

Hafner, D., Lillicrap, T., Norouzi, M., and Ba, J. (2020). Mastering Atari with Discrete World Models.

Hafner, D., Pasukonis, J., Ba, J., and Lillicrap, T. (2023). Mastering Diverse Domains through World Models.

Hansen, N., Lin, Y., Su, H., Wang, X., Kumar, V., and Rajeswaran, A. (2022). MoDem: Accelerating Visual Model-Based Reinforcement Learning with Demonstrations. In *The Eleventh International Conference on Learning Representations*.

Hartikainen, K., Geng, X., Haarnoja, T., and Levine, S. (2019). Dynamical Distance Learning for Semi-Supervised and Unsupervised Skill Discovery.

Henaff, M. (2019). Explicit Explore-Exploit Algorithms in Continuous State Spaces. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc.

Hu, E. S., Chang, R., Rybkin, O., and Jayaraman, D. (2023). PLANNING GOALS FOR EXPLORATION.

Janner, M., Fu, J., Zhang, M., and Levine, S. (2021). When to Trust Your Model: Model-Based Policy Optimization.

Kauvar, I., Doyle, C., Zhou, L., and Haber, N. (2023). Curious Replay for Model-based Adaptation.

Liu, M., Zhu, M., and Zhang, W. (2022). Goal-Conditioned Reinforcement Learning: Problems and Solutions.

Luo, Y., Xu, H., Li, Y., Tian, Y., Darrell, T., and Ma, T. (2021). Algorithmic Framework for Model-based Deep Reinforcement Learning with Theoretical Guarantees.

McCarthy, R., Wang, Q., and Redmond, S. J. (2021). Imaginary Hindsight Experience Replay: Curious Model-based Learning for Sparse Reward Tasks.

Mendonca, R., Rybkin, O., Daniilidis, K., Hafner, D., and Pathak, D. (2021). Discovering and Achieving Goals via World Models.

Micheli, V., Alonso, E., and Fleuret, F. (2022). Transformers are Sample-Efficient World Models. In *The Eleventh International Conference on Learning Representations*.

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.

Oudeyer, P.-Y., Kaplan, F., and Hafner, V. V. (2007). Intrinsic Motivation Systems for Autonomous Mental Development. *IEEE Transactions on Evolutionary Computation*, 11(2):265–286.

Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. (2017). Curiosity-driven Exploration by Self-supervised Prediction. In *Proceedings of the 34th International Conference on Machine Learning*, pages 2778–2787. PMLR.

Pislar, M., Szepesvari, D., Ostrovski, G., Borsa, D. L., and Schaul, T. (2022). When should agents explore? In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Pitis, S., Chan, H., Zhao, S., and Stadie, B. (2020). Maximum Entropy Gain Exploration for Long Horizon Multi-goal Reinforcement Learning.

Plappert, M., Andrychowicz, M., Ray, A., McGrew, B., Baker, B., Powell, G., Schneider, J., Tobin, J., Chociej, M., Welinder, P., Kumar, V., and Zaremba, W. (2018). Multi-Goal Reinforcement Learning: Challenging Robotics Environments and Request for Research.

Pong, V. H., Dalal, M., Lin, S., Nair, A., Bahl, S., and Levine, S. (2019). Skew-Fit: State-Covering Self-Supervised Reinforcement Learning.

Qiu, W., Mao, W., and Zhu, H. (2023). Instructing Goal-Conditioned Reinforcement Learning Agents with Temporal Logic Objectives. *Advances in Neural Information Processing Systems*, 36:39147–39175.

Qiu, W. and Zhu, H. (2021). Programmatic Reinforcement Learning without Oracles. In *International Conference on Learning Representations*.

Ren, Z., Dong, K., Zhou, Y., Liu, Q., and Peng, J. (2019). Exploration via Hindsight Goal Generation.

Sekar, R., Rybkin, O., Daniilidis, K., Abbeel, P., Hafner, D., and Pathak, D. (2020). Planning to Explore via Self-Supervised World Models. In *Proceedings of the 37th International Conference on Machine Learning*, pages 8583–8592. PMLR.

Shyam, P., Jaśkowski, W., and Gomez, F. (2019). Model-Based Active Exploration. In *Proceedings of the 36th International Conference on Machine Learning*, pages 5779–5788. PMLR.

Trott, A., Zheng, S., Xiong, C., and Socher, R. (2019). Keeping Your Distance: Solving Sparse Reward Tasks Using Self-Balancing Shaped Rewards.

Tuyls, J., Yao, S., Kakade, S. M., and Narasimhan, K. (2022). Multi-stage episodic control for strategic exploration in text games. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Wagenmaker, A., Shi, G., and Jamieson, K. (2023). Optimal Exploration for Model-Based RL in Nonlinear Systems. In *Thirty-Seventh Conference on Neural Information Processing Systems*.

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

Zhang, L., Yang, G., and Stadie, B. (2021). World Model as a Graph: Learning Latent Landmarks for Planning.

Zhang, W., Wang, G., Sun, J., Yuan, Y., and Huang, G. (2023). STORM: Efficient Stochastic Transformer based World Models for Reinforcement Learning. In *Thirty-Seventh Conference on Neural Information Processing Systems*.

Zhang, Y., Abbeel, P., and Pinto, L. (2020). Automatic Curriculum Learning through Value Disagreement.

Appendix

A Extended Related Work

In addition to reshaping the exploration reward function, goal-directed exploration represents a widely employed strategy that sets exploration goals distinct from the final task objective. Essentially, this approach aims to select goals that present challenges to the current policy while remaining achievable. Prior works have proposed various methods to generate goals for goal-directed exploration. (Zhang et al. (2020)) proposed to do automatic curriculum generation of goals based on the epistemic uncertainty of value functions. (Florensa et al. (2018)) use generative adversarial training to automatically generate goals, leveraging goal difficulty as a guiding factor. (Pong et al. (2019); Pitis et al. (2020)) proposed to use maximum entropy of achieved goal distribution to guide goal selection. (Ecoffet et al. (2019)) introduce a more efficient exploration methodology known as Go-Explore. This approach initially employs the goal-conditioned policy (Go-phase), followed by the rollout of the exploration policy from the terminal state of the goal-conditioned phase (Explore-phase). Go-Explore facilitates exploration initiation from a state area accessible by the current capabilities of the goal-conditioned policy.

PEG (Hu et al. (2023)) proposes computing the exploration potential by simulating Go-Explore trajectories using a world model to identify goals characterized by elevated average exploration rewards in the Explore-phase. This metric incorporates anticipated exploration rewards of the Explore-phase, providing an advantage for Go-Explore. However, 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)) directly in the observation space. L3P (Zhang et al. (2021)) employs temporal distance to train a latent space, facilitating clustering within this space to delineate key state areas based on reachability. Our approach proposes exploration from the periphery of these key state regions, aiming to balance exploration of unknown territories while constraining exploration starting points to the edges of key state regions, thus avoiding meaningless exploration from widely sampled point from observation space.

Model-based reinforcement learning (MBRL) has seen significant advancements in recent years, driven by the development of sophisticated world models and planning algorithms. One notable approach is Stochastic Ensemble Value Expansion (STEVE) (Buckman et al. (2019)), which enhances sample efficiency by leveraging ensemble models to reduce overfitting and uncertainty in value estimates. Similarly, the work by Chua et al. (Chua et al. (2018)) demonstrates that probabilistic dynamics models can be effectively used to achieve high performance in a small number of trials. In the realm of combining model-based and model-free methods, Deisenroth and Rasmussen (Deisenroth and Rasmussen (2011)) introduced PILCO, a data-efficient policy search method that uses Gaussian processes for dynamics modeling. More recent advancements include the integration of large pre-trained models for world model construction and task planning, as explored by Guan et al. (Guan et al. (2023)). While exploring how to increase the efficiency and capabilities of RL, researchers are also investigating how to ensure the stability and interpretability of artificial intelligence systems(Qiu et al. (2023); Cui and Zhu (2021); Qiu and Zhu (2021)). The Dreamer framework by Hafner et al. (Hafner et al. (2019a)) utilizes latent imagination to learn behaviors directly from pixel observations, and its extensions (Hafner et al. (2020, 2023)) have shown impressive results in mastering diverse domains. The Recurrent World Models by Ha and Schmidhuber (Ha and Schmidhuber (2018)) also contribute to this line of work by facilitating policy evolution through latent space planning. Several approaches focus on improving exploration strategies within MBRL. For instance, the use of cross-entropy methods for Monte-Carlo Tree Search (Chaslot et al. (2008)) and the Curious Replay mechanism (Kauvar et al. (2023)) have been proposed to enhance exploration efficiency. The work by Wagenmaker et al. (Wagenmaker et al. (2023)) further explores optimal exploration strategies in nonlinear systems. Additionally, transformers have been leveraged for their sample efficiency in world modeling (Micheli et al. (2022); Zhang et al. (2023)), demonstrating their potential in complex environments. The integration of demonstrations into visual model-based reinforcement learning, as seen in MoDem (Hansen et al. (2022)), showcases another avenue for improving learning efficiency. Luo et al. (Luo et al. (2021)) provide a comprehensive framework with theoretical guarantees, while Janner et al. (Janner et al. (2021)) address the critical question of when to trust the learned models.

B Extended Background

B.1 Dreamer World Model

We use the world model structure $\hat{M}$ of recurrent state-space model (RSSM) of Dreamer(Hafner et al. (2019a,b, 2020, 2023)) to learn the dynamics. The complete model state of the RSSM is the concatenation of deterministic states and stochastic states, with the latter being generated by the former. The deterministic state h_t can used to get the prior state $\hat{z}_t$ and posterior state z_t. The $\hat{z}_t$ aims to predict the posterior without access to the current input state x_t while the posterior state z_t is concluded by integrating the encoded information of current input state x_t. The deterministic state h_t is updated by the recurrent transition function f_ϕ using the concatenation (h_t, z_t) or $(h_t, \hat{z}_t)$ as input. The world model is summarized in Figure 8, and the formulas of components are shown in Equation 8:

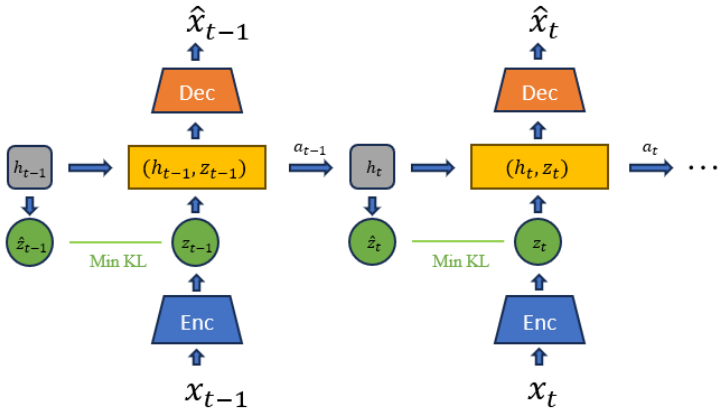


Figure 8: RSSM Structure

$$
\begin{aligned}
\text{Encoder:} \quad & e_t = f_E(e_t|x_t) \\
\text{Recurrent model:} \quad & h_t = f_\phi(h_{t-1}, z_{t-1}, a_{t-1}) \\
\text{Representation model:} \quad & z_t \sim q_\phi(z_t|h_t, e_t) \\
\text{Transition predictor:} \quad & \hat{z}_t \sim p_\phi(\hat{z}_t|h_t) \\
\text{Decoder:} \quad & \hat{x}_t \sim f_D(\hat{x}_t|h_t, z_t)
\end{aligned} \tag{8}
$$

B.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.

$$b_t = (h_t, z_t) \tag{9}$$

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.

$$\text{Predicted embedding:} \quad emb(b_t) = \hat{e}_t \approx e_t, \quad \text{where} \quad e_t = f_E(x_t) \tag{10}$$

$$\text{Temporal distance: } D_t(\hat{e}_t, \hat{e}_{t+k}) \approx k/H \quad \text{where} \quad \hat{e}_t = emb(b_t) \quad \hat{e}_{t+k} = emb(b_{t+k}) \tag{11}$$

$$r_t^G(b_t, b_{t+k}) = -D_t(\hat{e}_t, \hat{e}_{t+k}) \tag{12}$$

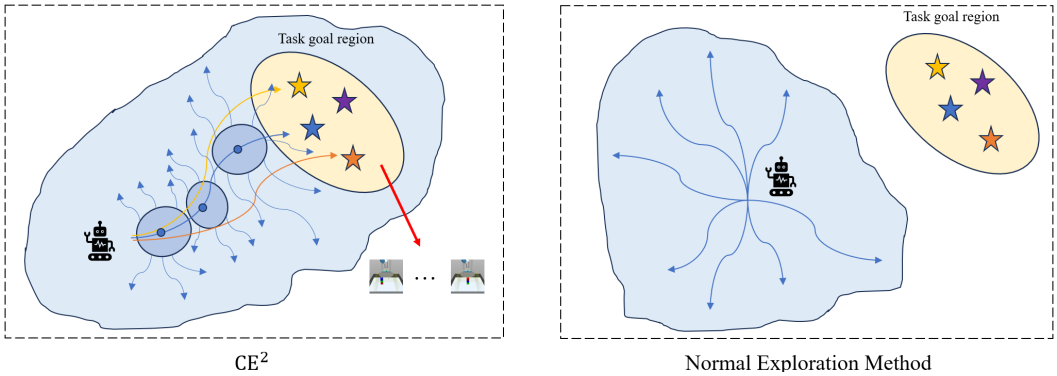


Figure 9: Illustration of differences between our mothod and other exploration methods.

C Limitations and Future Work

Our method clusters in the latent space, which necessitates a well-trained latent space. This latent space must not only accurately reconstruct the original state space and facilitate dynamic prediction but also reflect the reachable distances between different states. Therefore, training this latent space requires a temporal distance predictor that can accurately estimate the number of action steps needed between two states. We utilize the temporal distance predictor network from LEXA, which constructs intrinsic goal-conditioned rewards, and this network is trained using simulated trajectories. Compared to training the temporal distance predictor with real trajectories, using simulated trajectories offers greater stability. Our method requires the temporal distance predictor to reliably estimate the number of action steps needed to transition from one state to another, which is a crucial prerequisite for ensuring the effectiveness of CE^2 . Moreover, Although CE^2 has achieved remarkable success in tasks such as stacking blocks and rotating objects, there remain more challenging tasks that CE^2 needs to address. For instance, environments such as inserting a peg into a hole or fluid tasks in ManiSkill2(Gu et al. (2023)).

Besides, our realization of CE^2 is based on Dreamer, a model-based reinforcement learning(MBRL) agent known for its higher sample efficiency but greater computational demands compared to model-free alternatives. This increased resource requirement stems from the necessity to develop a world model. In CE^2 , this world model is utilized to train policies and value functions through simulated trajectories. At the same time, CE^2 use the PEG as the filter of exploration potential, which rely on world model to select goals that guide exploration. Creating a model-free version of CE^2 would simplify both its computational and conceptual aspects, a task we plan to undertake in future research.

D Environments

D.1 3-Block Stacking

In this task, a robot is required to stack three blocks into a tower. In PEG, evaluations are conducted on goals of varying difficulty levels, including 3 easy goals(picking up a single block), 6 medium goals(stacking two blocks), and 6 hard goals(stacking three blocks). We evaluate our agent solely on the 6 hard goals. At the same time, we use only 3 hard goals provided by the training environment as guiding goals for CE^2-G . Relying solely on the most challenging goals for training and evaluation presents a heightened challenge for both CE^2 and CE^2-G . However, we observed that CE^2 and CE^2-G are capable of spontaneously discovering additional easy and medium difficulty goals through clustering in latent space, as these serve as crucial transitional states towards the hard goals. The environment features 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 xyz movements of the gripper and the fourth dimension controlling the gripper's finger movement. The robot achieves success when the L2 distance between each block's xyz position and its target position is less than 3

cm. This environment is a modified version of the FetchStack3 environment from Pitis et al. (2020), incorporating adjustments to better test the robot's precision in stacking.

D.2 Walker

In this environment, a 2D walker robot is trained and evaluated the locomotion capabilities of on a flat surface. The environment code is sourced from Mendonca et al. (2021). In order to fully evaluate the agent's ability and accuracy to travel to longer distances, we increased the number of evaluation goals in PEG from 4($\pm 7, \pm 12$) to 12($\pm 13, \pm 16, \pm 19, \pm 22, \pm 25, \pm 28$) along the x axis from its initial position. Noting that, in CE^2-G , we only use goals at $\pm 13, \pm 16$ as the training goals returned by environments, but evaluate on all 12 goals. Success is determined by checking if the agent's x position falls within a small margin of the target x position. The state and goal space are nine-dimensional, encompassing the walker's xz positions and joint angles.

D.3 Ant Maze

This environment is adapted from the Ant Maze described in Pitis et al. (2020), with a little modifications. Like PEG, we set the goal space to be same with state space which including the ant's xyz positions along with joint positions and velocities, and an additional room was added in the top left to introduce a more challenging goal. In this complex environment, a high-dimensional ant robot must navigate from the bottom left to the top left of a maze, passing through hallways. The task is challenging due to its long duration, with episodes lasting 500 timesteps, and the considerable distance to be traversed. Compared to evaluation on goals both in top left room and in the central hallway in PEG, our evaluation only focuses on the ant reaching the most difficult four goals in the top left room. Besides, we use all 32 goals of different positions in the maze to be the training goals returned by environment for CE^2-G . The maze itself measures approximately 6 x 8 meters. Success is determined by ensuring the L2 distance between the ant's xy position and the goal is less than 1.0 meter, roughly the width of a cell in the maze. The Ant Maze environment features the highest dimensional state and goal spaces, totaling 29 dimensions. These include the ant's position, joint angles, and joint velocities. Specifically, the first three dimensions represent the xyz position, the next 12 dimensions correspond to the joint angles of the ant's four limbs, and the remaining 14 dimensions capture the velocities in the xy plane and of each joint. The action space consists of 8 dimensions, controlling the hip and ankle actuators of the ant's limbs.

D.4 Point Maze

The 2D point agent starts at the bottom left corner of a 10 x 10 maze and is tasked with reaching the top right corner within 50 timesteps. The state space and action space are both two-dimensional, corresponding to the agent's position and velocity on the plane. Success is determined if the L2 distance between the agent's position and the goal is less than 0.15. This environment is directly adapted from Pitis et al. (2020) without any modifications. In CE^2-G , the training goals from environments is randomly chose from 11 goals in different positions of maze.

D.5 Block and Pen Rotation

The hand must manipulate both a thin pen or a block to achieve target rotations. Manipulating the thin pen presents a greater challenge than manipulating the block due to the pen's tendency to slip, requiring more precise control. We utilize variant versions of the gymnasium environments: "HandManipulatePenRotate-v1" and "HandManipulateBlockRotateXYZ-v1". These environments introduce randomized target rotations for all axes of the block and x, y axes of the pen in each episode. The state space for both tasks consists of 61 dimensions, providing details on the robot's joint and object states, as well as goal information. The goal space remains consistent at 7 dimensions, indicating the target pose information. During evaluation, the latest policy is evaluated across 50 episodes for each task, with each episode featuring a distinct random goal. In CE^2-G , the training goals from environments is also randomly generated.

E Baselines

In this section, we present the pseudocode for all baseline methods. Note that, except for L3P, each baseline employs a different strategy for sampling data in the real environment within this framework. Therefore, we first display the general training framework for MBRL and subsequently provide the pseudocode for each baseline's data sampling method in the real environment.

Algorithm 4 General MBRL Training Framework

Input: Policy π^G, π^E, Environment Goal Distribution G, World Model $\hat{M}$, reward function r^G, r^E
$\mathcal{D} \leftarrow \{\}$ Initialize buffer.
for Episode $i = 1$ to N_{train} **do**
    $\tau \leftarrow$ Collect trajectories$(\dots)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \tau$
    Update model $\hat{M}$ with $(s_t, a_t, s_{t+1}) \sim \mathcal{D}$
    Update π^G in imagination with $\hat{M}$ to maximize r^G
    Update π^E in imagination with $\hat{M}$ to maximize r^E

E.1 Go-Explore

To enhance the exploration area of the explorer, we adopt the Go-Explore strategy (Ecoffet et al. (2019)). This approach initially employs a goal-conditioned policy, π^G, to approach a specified goal g as closely as possible, a process referred to as the Go-phase. Following this, the explorer, π^E, is used to further explore the environment starting from the terminal state of the Go-phase, known as the Explore-phase.

The effectiveness of the trajectories generated by the Go-Explore strategy heavily depends on the choice of the goal g during the Go-phase. Thus, establishing an efficient goal selection mechanism for the Go-phase is crucial. If the chosen goal g is too easy, the explorer will not sufficiently explore the environment. Conversely, if the goal g is too difficult, the goal-reaching policy π^G will be unable to approach it effectively. Therefore, the objective is to develop a goal selection mechanism that identifies a goal g capable of guiding the agent to a region with high exploration potential during the Go-phase. Below, we provide the pseudocode for the Go-Explore strategy.

Algorithm 5 Go Explore Framework

function GO-EXPLORE(g, π^G, π^E)
    $s_0 \leftarrow$ env.reset()
    $\tau \leftarrow \{s_0\}$
    for Step $t = 1$ *to* T_{Go} **do**
        $s_t \leftarrow$ env.step($\pi^G(s_{t-1}, g)$)
        $\tau \leftarrow \tau \cup \{s_t\}$
        if agent reach g **then**
            break
    $t_e = t$
    for Step $t = t_e$ *to* $t_e + T_{\text{Explore}}$ **do**
        $s_t \leftarrow$ env.step($\pi^E(s_{t-1})$)
        $\tau \leftarrow \tau \cup \{s_t\}$
return τ

E.2 GC-Dreamer

GC-Dreamer follows a goal-conditioned approach where trajectories are collected by goal-conditioned policy π^G, and the goals are returned from the training environment.

Algorithm 6 GC-Dreamer Goal Sampling

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

E.3 PEG

PEG adopts a strategy where trajectories are collected by optimizing a specific equation using the MPPI method. The optimized goal is then used to guide exploration through the GO-EXPLORE algorithm.

Algorithm 7 PEG Sampling

function COLLECT TRAJECTORIES(...)
    $g \leftarrow$ Optimize Equation 7 with MPPI
    $\tau \leftarrow$ *GO-EXPLORE*(g, π^G, π^E)
return τ

E.4 PEG-G

PEG-G combines the utilization of goals from the environment with those generated by optimizing the equation using MPPI. This approach alternates between the two strategies based on the episode index.

Algorithm 8 PEG-G Sampling

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

E.5 MEGA

For model-based MEGA, we directly utilize the implementation method described in the PEG paper. This involves transplanting MEGA's KDE model and using a goal-conditioned value function within the LEXA framework to filter goals based on reachability. The PEG paper has demonstrated that their implementation of MEGA achieves superior performance compared to the original MEGA baseline.

Algorithm 9 MEGA Goal Sampling

function COLLECT TRAJECTORIES(...)
    $g \leftarrow \min_{g \in \mathcal{D}} \widehat{p}(g)$
    $\tau \leftarrow$ *GO-EXPLORE*(g, π^G, π^E)
return τ

E.6 MEGA-G

Similar to PEG-G, MEGA-G alternates between using goals from the environment and MEGA goal picking strategy.

Algorithm 10 MEGA-G Goal Sampling

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

E.7 MEGA+PEG

MEGA+PEG combines the strategies of MEGA and PEG. this baseline firstly employs MEGA to sample a batch of candidate goals, all of which have low density in the replay buffer. Subsequently, their exploration potential is evaluated using PEG, with the most valuable one selected as the exploration goal.

Algorithm 11 MEGA+PEG Goal Sampling

function COLLECT TRAJECTORIES(...)
  $G \leftarrow$ Top-10 smallest $\widehat{p}(g)$ for $g \in \mathcal{D}$
  $g \leftarrow$ Optimize Equation 7 for $g \in G$
  $\tau \leftarrow$ *GO-EXPLORE*(g, π^G, π^E)
return τ

E.8 CE^2-noPEG

CE^2-noPEG utilizes a goal-picking strategy based on Gaussian Mixture Model (GMM) clustering to generate goals for exploration without employing PEG optimization. The goal picked by this method is sampled at the edge of our latent space clusters, which is the main contribution of our paper.

Algorithm 12 CE^2-noPEG Goal Sampling

function COLLECT TRAJECTORIES(...)
  $D_{exp} \leftarrow \{\}$
  for episode $i = 1$ to N_τ **do**
    $G_{candidate} \leftarrow$ Sample $N_{candidate}$ points from GMM
    $G_{edge} \leftarrow N_{edge}$ points in $G_{candidate}$ with the smallest total probability of the GMM.
    $g^E \leftarrow$ Randomly select a g from G_{edge}
    $\tau \leftarrow$ *GO-EXPLORE*(g^E, π^G, π^E)
return τ

E.9 L3P

Our implementation of L3P follows the original code provided in the L3P paper(Zhang et al. (2021)). For more details on the pseudocode and specific implementation, please refer to the descriptions in their paper.

F Implementation Details

F.1 Farthest Point Sampling (FPS) Algorithm

Algorithm 13 Farthest Point Sampling (FPS)

```
function FPS(points, num_samples)
    sampled_points ← [ ]
    first_point ← random.choice(points)
    sampled_points.append(first_point)
    min_distances ← [float('inf')] × len(points)
    for each point p in points do
        min_distances[p] ← distance(p, first_point)
    for iteration i = 1 to num_samples-1 do
        farthest_point_index ← argmax(min_distances)
        farthest_point ← points[farthest_point_index]
        sampled_points.append(farthest_point)
        for each point p in points do
            min_distances[p] ← min(min_distances[p], distance(p, farthest_point))
    return sampled_points
```

The Farthest Point Selection (FPS) algorithm, commonly employed in various applications including point cloud simplification and image sampling, initializes by creating an empty list termed 'sampled_points' to retain the selected points. The process initiates by randomly selecting an initial point from the input point set, designated as 'points', and appending it to 'sampled_points'. Subsequently, 'min_distances' is initialized to track the minimum distance from each point to any of the sampled points, with initial values set to infinity. The core procedure entails iteratively selecting points until reaching the desired number of samples. 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. The algorithm incorporates two auxiliary functions: 'distance(point1, point2)', facilitating the computation of the Euclidean distance between two points, and 'argmax(array)', which returns the index of the maximum value within an array. See pseudocode Algorithm 13 for more details about FPS.

F.2 Runtime

Table 1: Runtimes per experiment.

	Total Runtime (Hours)	Total Steps	Episode Length	Seconds per Episode
3-Block Stacking	60	1e6	150	31.34
Walker	36	1e6	150	18.62
Ant Maze	56	1e6	500	96.91
Point Maze	36	1e6	50	5.60
Block Rotation	58	1e6	150	30.74
Pen Rotation	58	1e6	150	30.42

We conduct each experiment on GPU Nvidia A100 and require about 5GB of GPU memory. See Table 1 for specific running time of CE^2 for different task. The running time of CE^2-G has no big difference with CE^2 . The neural network updates of the policies and world model take most of runtime. However, CE^2 takes more time in goal selection compared to PEG and MEGA. This is because CE^2 need evaluate the exploration potential of candidate goals sampled at the edge of latent clusters every time it picks a goal for exploration. In the PEG algorithm, the MPPI parameters are only updated at fixed intervals of multiple episodes. This means that the exploration potential is assessed for a batch of data after a certain number of episodes have been completed. In contrast, our method evaluates the exploration potential for a batch of candidate goals at each episode when

sampling exploration goals. However, we can also follow the PEG by evaluating the exploration potential of a batch of candidate goals after multiple episodes. Between these updates, we can select exploration goals from the previously evaluated set of candidate goals. While this may sacrifice some algorithm performance, it can significantly reduce the time CE^2 requires to select goals.

F.3 Hyperparameters

Similar to PEG, 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. For the Gaussian Mixture Model(GMM), we set the learning rate to be 3e-4, optimizer to be Adam. We also test result of different cluster number(10, 30, 50) setting as we show in ablation experiment. Every time we sample points in the GMM, we set the point number $N_{candidate} = 1000$ and we set the edge point number $N_{edge} = 100$. After then, we evaluate the exploration potential of these 100 points and pick the point as goal state which have largest exploration potential value. In addition, to improve clustering in the latent space, we reduced the latent space dimension from 400(LEXA setting) to 50. We tested the training speed and results with both a latent space dimension of 400 and 50. We found that a latent space dimension of 50 allows for faster clustering and accelerates the training process. Meanwhile, although a latent space dimension of 400 reduces the training speed a little, it does not affect the final success rate.

G Additional Experiments

G.1 Space explored image for 3-Block Stacking

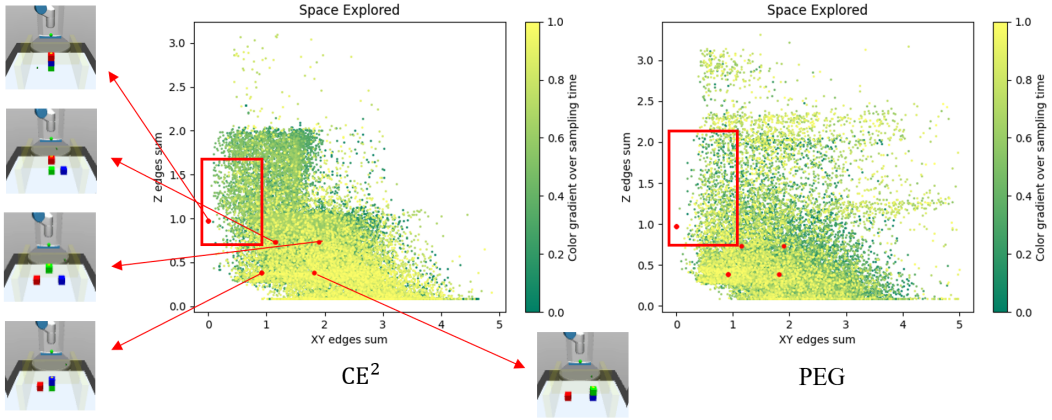


Figure 10: Space explored by CE^2 and PEG in the 3-Block Stacking environment at 1M steps. X-axis: the sum of the three sides of the triangle projected on the x-y plane by the three block-connected triangles. Y-axis: sum of heights (z-coordinates) of the three blocks. Red points: evaluation goals. Other points: observations of trajectories sampled in real environment. Color from green to yellow means to be sampled more recent.

In 3-Block Stacking, we innovatively designed a method based on the coordinates of three blocks to demonstrate the degree of exploration in the environment, showed in Figure 10. Firstly, we establish connections between the coordinates of three blocks in three-dimensional space, forming a spatial triangle. This spatial triangle serves to express the relative positions and distances of the three blocks within the space. Subsequently, we project this spatial triangle onto the xy-plane. The summation of the lengths of the sides of the projected triangle on the xy-plane reflects the dispersal of the three blocks within the xy-plane, while the total sum of the z-coordinates of the three blocks indicates their relative positions in height Utilizing the former as the x-axis and the latter as the y-axis, we depict a schematic illustration of the spatial exploration of 3-Block Stacking. We observe that CE^2 exhibits a more targeted and in-depth exploration around the target space (highlighted within the red box) compared to PEG. Simultaneously, we observed that PEG tends to conduct numerous explorations in the upper-left region of the exploration graph, which are often futile and irrelevant to the goal of

completing block stacking. This highlights the advantage of CE^2 , which benefits from the constraints imposed by clustering and avoids blindly exploring areas. Moreover, sampling at the edges of clusters ensures the profitability of exploration, enabling more efficient exploration of the vicinity of the goal space compared to PEG.

G.2 More Exploration Process

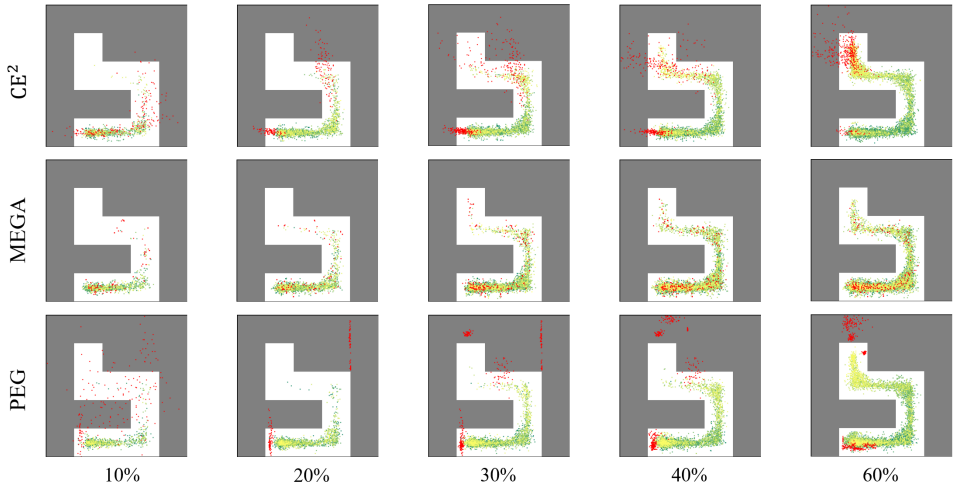


Figure 11: Comparison of exploration goals generated by CE^2 , MEGA and PEG

We present a comparison of exploration targets generated by CE^2 , MEGA and PEG approaches over the training process in the Ant Maze environment. In the Fig 11, red points represent the generated exploration targets by different methods. We observe that the exploration targets generated by CE^2 are significantly superior to those generated by MEGA and PEG. Specifically, CE^2 consistently generates points located at the forefront of agent exploration and within the agent's reachable capability range. In contrast, the targets generated by MEGA exhibit greater dispersion and sparsity, which are disadvantageous for concentrated exploration of forefront regions. Moreover, PEG consistently generates targets outside the Maze channels, rendering these exploration targets not only far beyond the agent's capability range but also meaningless.

G.3 Centroids Visualization

By decoding the centroids of GMMs in latent space, we can visualize some centroids of GMMs in CE^2 , showing in Fig. 12

G.4 Full results for CE^2-G

Please see Fig. 13

G.5 More Ablation Experiments

We conducted experiments in the CE^2 with different numbers (10, 30, 50) of clusters to observe if the results are sensitive to the number of clusters. The results are shown in Figure 14. We found that the performance of CE^2 is not strongly correlated with the number of clusters; as long as the number of clusters is sufficient to represent key state regions, the results tend to be stable, which demonstrates the robustness of our approach.

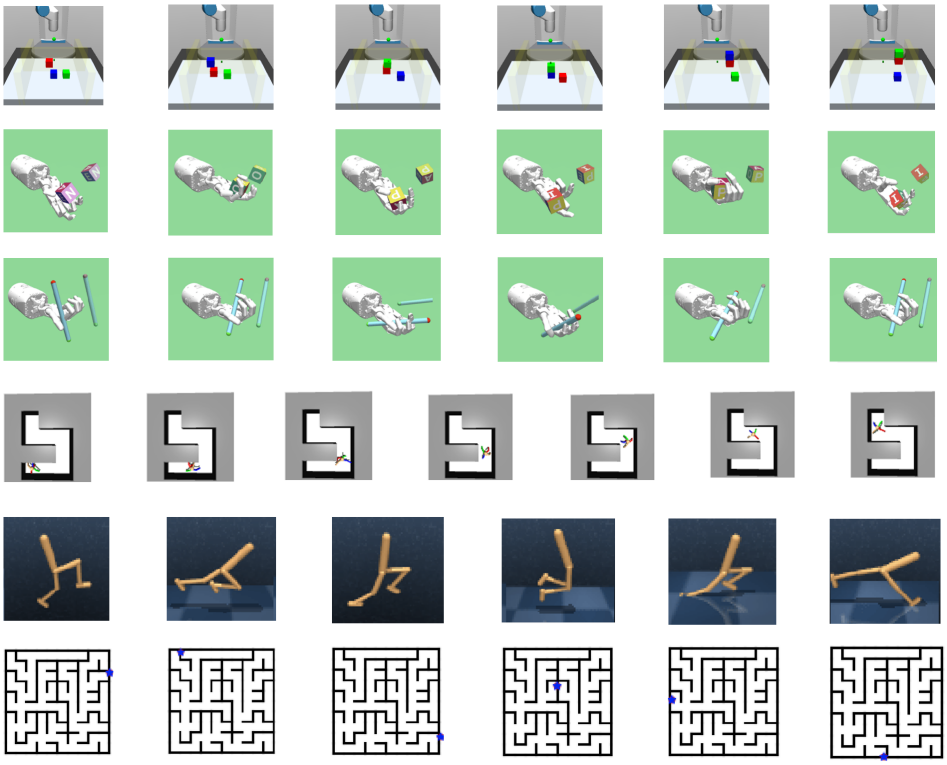
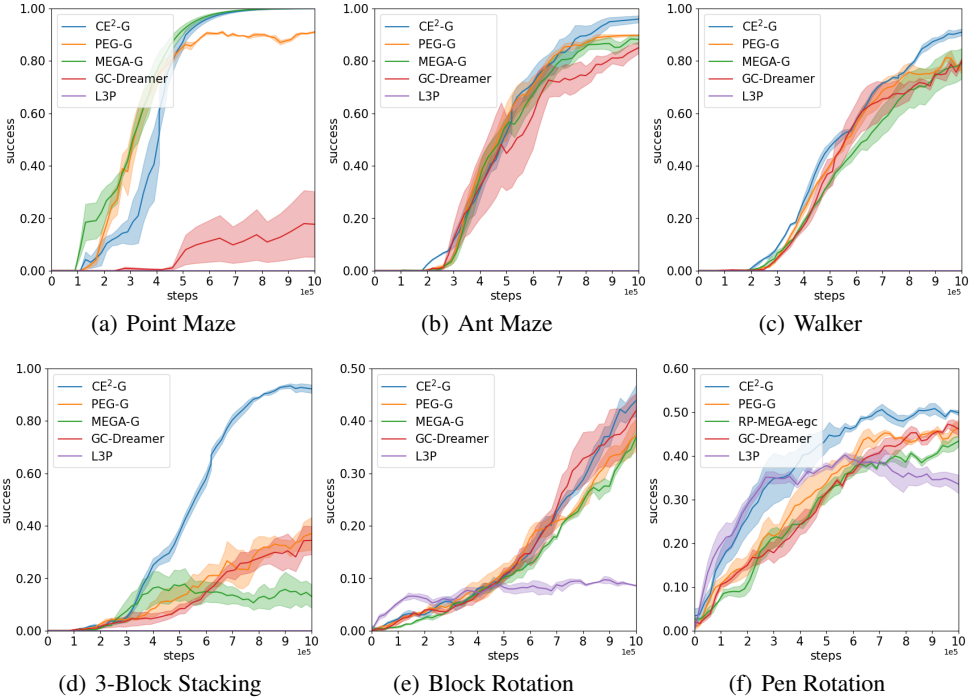


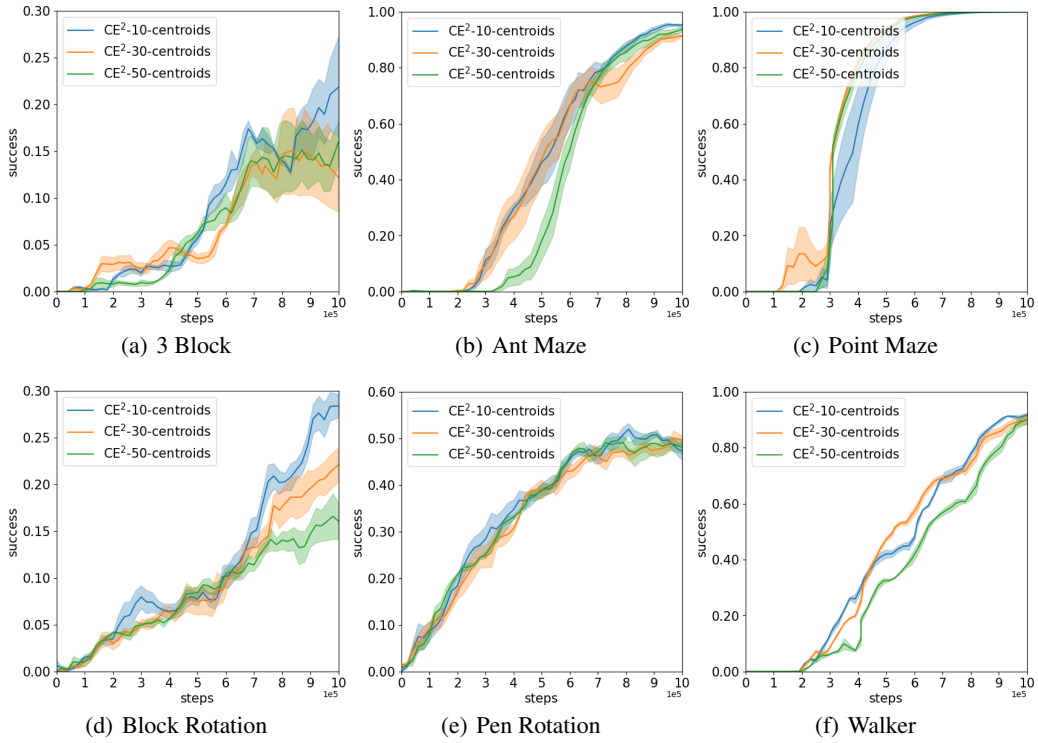
Figure 12: some centroids visualization of GMMs in CE^2 .



(a) Point Maze (b) Ant Maze (c) Walker

(d) 3-Block Stacking (e) Block Rotation (f) Pen Rotation

Figure 13: Full experiment Results comparing CE^2-G with the baselines in six environments.



(a) 3 Block (b) Ant Maze (c) Point Maze

(d) Block Rotation (e) Pen Rotation (f) Walker

Figure 14: Ablation Study with Different Cluster Number.