Go Beyond Imagination: Maximizing Episodic Reachability with World Models

Yao Fu [1] **Run Peng** [1] **Honglak Lee** [1 2]

Abstract

Efficient exploration is a challenging topic in reinforcement learning, especially for sparse reward tasks. To deal with the reward sparsity, people commonly apply intrinsic rewards to motivate agents to explore the state space efficiently. In this paper, we introduce a new intrinsic reward design called GoBI - Go Beyond Imagination, which combines the traditional lifelong novelty motivation with an episodic intrinsic reward that is designed to maximize the stepwise reachability expansion. More specifically, we apply learned world models to generate predicted future states with random actions. States with more unique predictions that are not in episodic memory are assigned high intrinsic rewards. Our method greatly outperforms previous state-of-the-art methods on 12 of the most challenging Minigrid navigation tasks and improves the sample efficiency on locomotion tasks from DeepMind Control Suite.

1. Introduction

Efficient exploration in state space is a fundamental challenge in reinforcement learning (RL) (Hazan et al., 2019; Lee et al., 2019), especially when the environment rewards are sparse (Mnih et al., 2013; 2016; Schulman et al., 2017) or absent (Liu & Abbeel, 2021; Parisi et al., 2021). Such reward sparsity makes RL algorithms easy to fail due to the lack of useful signals for policy update (Riedmiller et al., 2018; Florensa et al., 2018; Sekar et al., 2020). A common approach for exploration is to introduce self-motivated intrinsic rewards such as state visitation counts (Strehl & Littman, 2008; Kolter & Ng, 2009) and prediction errors (Stadie et al., 2015; Pathak et al., 2017; Burda et al., 2018). Most of these intrinsic reward designs measure lifelong state novelty and prioritize visiting states that are less visited starting from the beginning of training.

While the above methods achieves great improvement on hard-exploration tasks like Montezuma's Revenge (Burda et al., 2018), they generally only work well on "singleton" environments, where training and evaluation environments are the same. However, due to the poor generalization performance of reinforcement learning in unseen environments (Kirk et al., 2021), nowadays researchers have been paying more attention on procedurally-generated environments (Cobbe et al., 2019; 2020; Flet-Berliac et al., 2021), where the nature of task remains the same but the environment is randomly constructed for each new episode. For example, a maze-like environment will have different maze structures, making it rare for the agent to encounter the same observations across different episodes. Therefore, lifelong novelty intrinsic motivations usually fail in hard procedurally-generated environments of this kind (Raileanu & Rocktäschel, 2020; Zha et al., 2021) because an agent will be trapped around newly-generated states.

Inspired by human's frequent use of short-term memory (Andersen et al., 2006; Eichenbaum, 2017) to avoid repeatedly visiting the same space, recent work propose to derive intrinsic rewards on episodic level (Savinov et al., 2018; Badia et al., 2020; Raileanu & Rocktäschel, 2020; Zha et al., 2021; Zhang et al., 2021). The episodic intrinsic rewards generally give bonus to large episodic-level state space visitation coverage, therefore encourage visiting as many states as possible in the same episode. However, *does visiting more states necessarily mean efficient episodic-level exploration*? We notice that some state visitations are unnecessary and can be avoided if they are predictable from episodic memory. For example, when navigating through a house to find a fridge, if you open a door and find an empty room, you do not need to go into it anymore because you can easily predict what the states are like in the room (i.e., intuitively speaking, you would be moving around in an empty room). With this inspiration, we propose to design the episodic intrinsic reward to not only maximize the number of visited states in an episode, but also consider those states that are not visited but can be predicted from episodic memory.

More precisely, we maintain an episodic buffer to store all the visited states as well as states reachable from the visited states within a few time steps. To get the reachable states,

[1]University of Michigan [2]LG AI. Correspondence to: Yao Fu <violetfy@umich.edu>, Run Peng <roihn@umich.edu>, Honglak Lee <honglak@eecs.umich.edu & honglak@lgresearch.ai>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.

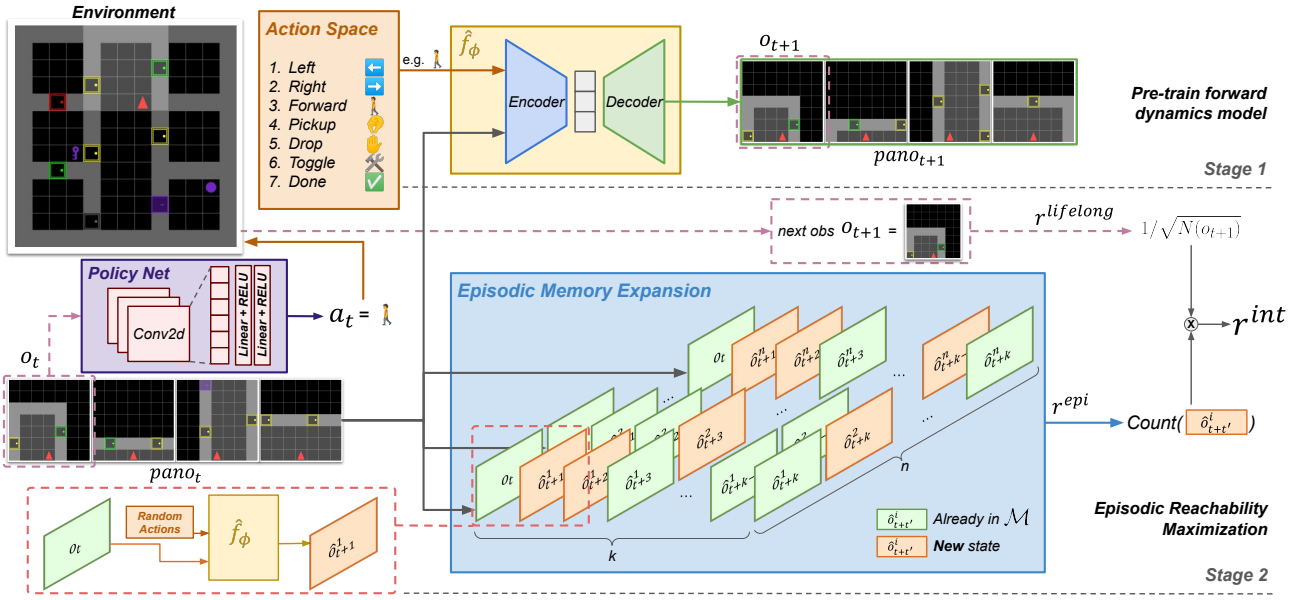


Figure 1. Illustration of how GoBI works on Minigrid. For the environment in the upper-left corner, the red triangle indicates the position and orientation of the agent. It has a 7×7 partially-observable view (highlighted). During pre-training (Stage 1), we collect data using a random policy to train a forward dynamics model $\hat{f}_\phi(pano_t, a_t) = o_{t+1}$, where $pano_t$ denotes the panoramic view as is defined in Section 3.1. For policy training (Stage 2), we apply $\hat{f}$ to predict observations in the future k time steps with n random actions for each step. We add the new ones to an episodic buffer $\mathcal{M}$ and take the change of size of $\mathcal{M}$ as the episodic intrinsic reward r^{epi}. The lifelong intrinsic reward is COUNT-based. Our intrinsic reward GoBI is $r^{\text{lifelong}} * r^{\text{epi}}$.

we train a world model with forward dynamics function and apply random actions to the learned dynamics model to predict future states. The predictions are added to the episodic buffer if they are not there already. We use the change of size of this episodic buffer as the episodic intrinsic reward. Following many previous work, we weight the episodic intrinsic reward by a lifelong intrinsic reward (Badia et al., 2020; Zhang et al., 2021) like the COUNT-based rewards. With this newly proposed intrinsic reward design **GoBI - Go Beyond Imagination**, the agent is expected to both explore the most of the state space throughout training to discover extrinsic rewards, and learn to act in an efficient manner within a single episode to avoid being trapped by seemingly novel states.

The contributions of this work can be highlighted as follows: (i) We propose a novel way to combine world models with episodic memory to formulate an effective episodic intrinsic reward design. (ii) In sparse-reward procedurally-generated Minigrid environments (Chevalier-Boisvert et al., 2018b), GoBI greatly improves the training sample efficiency in comparison with prior state-of-the-art intrinsic reward functions. (iii) GoBI extends well to DeepMind Control Suite (Tunyasuvunakool et al., 2020) with high-dimensional visual inputs and shows promising results on sparse-reward continuous control tasks. (iv) We analyze the design of GoBI and present extensive ablations to show the contribution of each component.

2. Method

We consider reinforcement learning problems framed as Markov Decision Process (MDP) $M = (\mathcal{S}, \mathcal{A}, T, R, \gamma)$, where $\mathcal{S}$ and $\mathcal{A}$ denote the state space and action space. $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the state transition function. $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function. γ is the reward discount factor. At each step t, the state of the environment is denoted as $s_t \in \mathcal{S}$. The agent generates an action $a_t \in \mathcal{A}$ to interact with the environment. The environment then transits to the next underlying state $s_{t+1} \in \mathcal{S}$. Apart from the new state s_{t+1}, the environment also returns an extrinsic reward r^{ext} that describes how well the agent reacts to s_t. In sparse-reward tasks, r^{ext} is usually 0. In this work, we follow the previous work to train RL algorithms with $r^{\text{ext}} + \lambda * r_t^{\text{int}}$, where r_t^{int} is a self-motivated intrinsic reward and λ is a hyper-parameter that controls the relative importance between intrinsic and extrinsic rewards.

2.1. Go Beyond Imagination

Reachable States and Episodic Buffer Our intrinsic reward design aims to exploit the information hidden inside the neighbourhood of states. We define a state A to be k-step *reachable* from state B if the agent can reach A from B within k time steps. During the training process, for each new episode, we initialize an empty episodic memory buffer $\mathcal{M}$. At time step t, we hash s_t as well as all the states

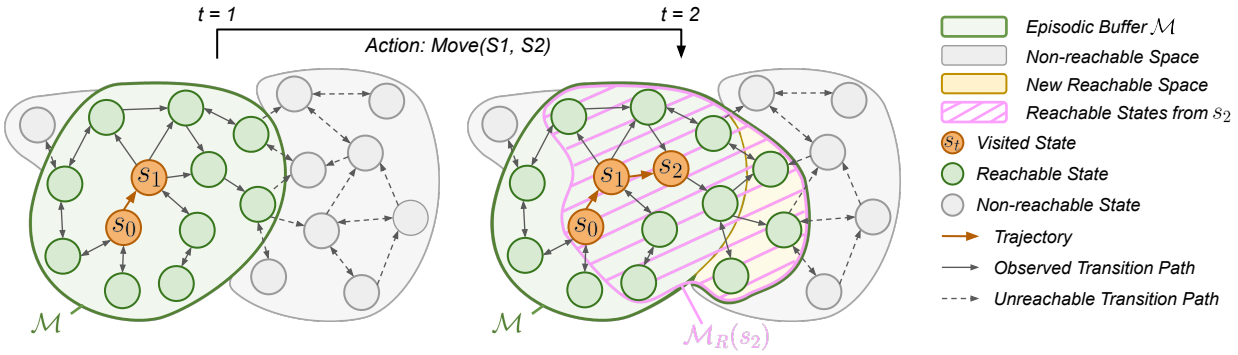


Figure 2. An illustration of how our episodic buffer updates. We consider reachable states that are $k = 2$ time steps away from s_t. After the agent moves from s_1 to s_2, the episodic buffer $\mathcal{M}$ (shaded in green) expands by 3 new reachable states (shaded in yellow). More formally, $\mathcal{M} \leftarrow \mathcal{M} \cup \mathcal{M}_R(s_2)$, where $\mathcal{M}_R(s_2)$ indicates a set of all the states 2-step reachable from s_2. Notice that all the states in the trajectory, i.e., s_0, s_1, s_2 are also added to the buffer.

reachable from s_t. We denote the set containing all the hash codes of s_t and its reachable states as $\mathcal{M}_R(s_t)$. Then we update $\mathcal{M}$ by $\mathcal{M} \leftarrow \mathcal{M} \cup \mathcal{M}_R(s_t)$. Storing the hash codes instead of directly storing the states may alleviate the potential memory issue of the buffer. We illustrate this process in Figure 2. When the agent reaches state s_2, we add 3 more states that are reachable from s_2 but not in $\mathcal{M}$.

Forward Dynamics In real environments, it is common that we do not have access to the neighbourhood relationship between states. However, we can learn a world model by training a forward dynamics model $\hat{f}_\phi(s_t, a_t) = s_{t+1}$ to predict the states reachable from s_t. This forward dynamics can be pre-trained using data collected by a random policy or trained online together with policy training. When training the policy, for each time step t, we generate $k \cdot n$ random actions and use the learned dynamics $\hat{f}_\phi$ to predict states in the future k steps. We hash the current state s_t as well as the predicted future states $\hat{s}^1_{t+1}, ..., \hat{s}^n_{t+1}, ..., \hat{s}^1_{t+k}, ..., \hat{s}^n_{t+k}$ and add the hash codes to the episodic buffer $\mathcal{M}$ if they are not in the buffer. Apart from alleviating potential memory issue as is mentioned in the last paragraph, using a hash function may also mitigate the noise introduced by $\hat{f}$. With a learned dynamics model, the predictions of reachable states are usually not perfect. However, in the experiment section we show that even with imperfect predictions, our method can improve the training sample efficiency a lot.

Episodic Novelty We aim to design an episodic-level novelty reward that guides the agent to extend the frontier of its predicted reachable space efficiently to discover states not visited and not predictable within the same episode. More specifically, we denote the size of the episodic buffer $\mathcal{M}$ as m_t at time step t and design a reachability-based bonus $r^{epi} = m_{t+1} - m_t$ that encourages the agent to find unexplored regions. For each time step, the agent is expected to reach the state that is reachable to more new states in the current episode.

Intrinsic Reward Formulation We further weight our episodic intrinsic reward by a lifelong intrinsic reward to encourage the agent to explore the regions that are not well explored in the past. More formally, the proposed intrinsic reward GoBI is defined as:

$$r^{\text{int}}_t = (m_{t+1} - m_t) * r^{\text{lifelong}}_t \tag{1}$$

Here, r^{lifelong}_t denotes lifelong intrinsic reward. We note that our framework is compatible with any choice of lifelong intrinsic reward. Specifically, we use the simple COUNT-based reward $1/\sqrt{N(s_{t+1})}$ for the navigation experiments on Minigrid environments (Chevalier-Boisvert et al., 2018b), where N denotes the count of s_{t+1} from the start of training.[1] For the experiments on DeepMind Control Suite (Tunyasuvunakool et al., 2020) we use the state-of-the-art intrinsic reward RE3 (Seo et al., 2021), which estimates state entropy by a random encoder.

Intrinsic Decay Intrinsic rewards are expected to be asymptotically consistent so that it will not influence the policy learning at later stage of training and result in a sub-optimal policy. To guarantee that the policy learning focuses more on extrinsic rewards as training proceeds, in RE3 (Seo et al., 2021), the authors apply exponential decay schedule for the intrinsic rewards to decrease over time. Although COUNT-based reward theoretically converges to 0 with enough exploration, it decreases quite slowly in procedurally-generated environments. Therefore, we also apply intrinsic reward decay when calculating GoBI by decreasing the intrinsic reward coefficient λ during training. We summarize our method in Algorithm 1 and illustrate the training process on Minigrid navigation tasks in Figure 1.

[1] For environments that are partially observable (e.g., in Minigrid, the agent observes a 7×7 pixel local view of the environment), we substitute state s_t with observation o_t when calculating the intrinsic rewards.

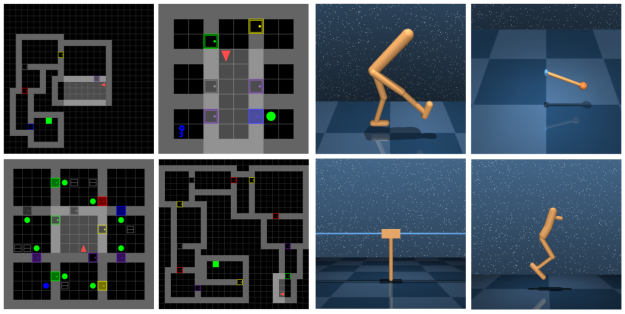
Algorithm 1 Go Beyond Imagination

Input: Intrinsic Reward Coefficient λ_0, Forward Prediction Step k, Number of Random Actions n, Intrinsic Reward Decay Parameter ρ
Initialize policy π_θ, dynamics model $\hat{f}_\phi$, replay buffer $\mathcal{B}$.
(Optional) Collect episodes with π_θ and train $\hat{f}_\phi$ with prediction loss
for episode $e = 1, 2, ...$ until convergence **do**
  Initialize episodic buffer $\mathcal{M}$.
  $\lambda \leftarrow \lambda_0 * (1-\rho)^{(e-1)*T}$
  for $t = 1$ **to** T **do**
    Execute π_θ in the environment to get a transition pair $(s_t, a_t, s_{t+1}, r_t^{\text{ext}})$.
    $m_t \leftarrow size(\mathcal{M})$
    $\mathcal{M} \leftarrow \mathcal{M} \cup \{hash(s_t)\})$
    for $t' = 1$ **to** k **do**
      generate n random actions $a_{t'}^1, ..., a_{t'}^n$
      $\mathcal{M} \leftarrow \mathcal{M} \cup \{hash(\hat{f}_\phi(\hat{s}_{t+t'}^i, a_{t'}^i) | i = 1, ..., n\})$
    end for
    $r_t^{\text{int}} = (size(\mathcal{M}) - m_t) * r^{\text{lifelong}}$
    $\mathcal{B} \leftarrow \mathcal{B} \cup \{(s_t, a_t, s_{t+1}, r_t^{\text{ext}} + \lambda * r_t^{\text{int}})\}$
  end for
  update π_θ with RL objective
  update $\hat{f}_\phi$ with prediction loss
end for

2.2. Conceptual Advantage of GoBI over Prior Works

Previous works including RIDE (Raileanu & Rocktäschel, 2020) and NovelD (Zhang et al., 2021) also combine episodic intrinsic reward with lifelong novelty as we do. However, most of them focus on episodic-level state visitation. For example, NovelD only assigns non-zero rewards to a state when it is visited for the first time in the episode. However, we notice that not all state visitations are necessary. The agent's goal for exploration is to gather information about the states. Therefore for states that are easily predictable from episodic memory, visiting them may not really help to acquire more information about the environment. In Figure 4, we plot the visitation heatmap of GoBI and NovelD to demonstrate the different exploration behaviours of the two methods.

Our method is closely related to another work that measures episodic curiosity (EC) (Savinov et al., 2018). In EC, the authors train a reachability network that takes in two arbitrary states and outputs a similarity score between 0 and 1, where 1 indicates the two states are the same and 0 indicates they are totally different. The network is trained using collected episodes by marking temporally close states as positive examples and temporally far ones as negative samples. Meanwhile, they also maintain an episodic buffer. A state s_t is compared with all the states in the buffer and gets a high intrinsic reward if the corresponding similarity scores are low. Only with low enough similarity scores do they add s_t to the buffer. Although their method and ours are similar at high level, they are different by design. For example, for an agent standing in front of an empty blind alley with dead end, agent trained with GoBI does not benefit in going deep into the blind alley because everything there can be predicted as reachable and added to the episodic buffer already. However, EC encourages going to the very end of the blind alley to reach the state with low similarity score and high intrinsic reward, even though going into an empty blind alley is not beneficial for exploration and wastes time that can be used to explore other parts of the environment. In Appendix C, we present the visitation heatmaps of policies learned by EC and find that it prefers going to the room corners, which well matches our explanation above.



(a) MiniGrid (b) Deepmind Control

Figure 3. Rendering of the environments used in this work. Left: 2D grid world navigation tasks that require object interactions. Right: DeepMind Control tasks with visual observations.

3. Experiments

In this section, we evaluate GoBI in two domains: 2D procedurally-generated Minigrid environments (Chevalier-Boisvert et al., 2018a) with hard-exploration tasks and locomotion tasks from DeepMind Control Suite (Tunyasuvunakool et al., 2020). The experiments are designed to answer the following research questions: (1) How does GoBI perform against previous state-of-the-art intrinsic reward designs in terms of training-time sample efficiency on challenging procedurally-generated environments? (2) Can GoBI successfully extend to complex continuous domains with high-dimensional observations, for example control tasks with visual observations? (3) How does each component of our intrinsic reward contribute to the performance? (4) What is the influence of the accuracy of the learned world models to our method?

3.1. Minigrid Navigation Tasks

Minigrid Environments MiniGrid (Chevalier-Boisvert et al., 2018a) is a set of partially-observable procedurally-

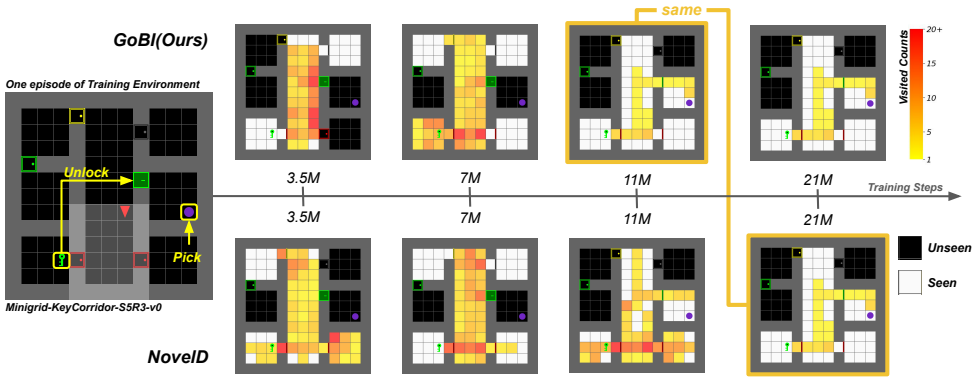


Figure 4. Visitation heatmaps on KeyCorridorS5R3 at different training stages. This figure compares the policy behaviour of GoBI and NovelD. A dark red color means plentiful visitations, white means the agent has seen the space but did not step on it, and black means space that are not discovered. It is worth noticing that early in the training (3.5M and 7M time steps), our policy already learns not to go into an empty room, likely because states in an empty room are easily predictable. On the contrary, even after 11M steps, an agent trained with NovelD still goes into an empty room (bottom-right corner) for more state visitations.

generated grid world navigation tasks. The agent is expected to interact with objects such as keys, balls, doors, and boxes to navigate through rooms and find the goal that is randomly placed in one of the rooms. The tasks only provide one sparse reward at the end of each episode, which indicates if the agent successfully finds the goal or not and how many steps it takes to reach the goal. In this work, we consider 3 types of tasks including MultiRoom, KeyCorridor, and ObstructedMaze. Some environments that we experiment on in this paper are shown in Figure 3a. The upper-right is a KeyCorridor-S4R3 environment, where the agent should learn to open the doors to find a key, use it to open the locked blue door, and pick up the green ball. The bottom-left figure shows an ObstructedMaze-Full environment, which is similar to KeyCorridor but more challenging. The rooms are larger, the doors are blocked by balls, and the keys are hidden in boxes. The upper-left and bottom-right environments are MultiRoom environments, in which the agent has to navigate through connected rooms to reach the goal in the last room.

Baselines We compare with state-of-the-art intrinsic reward designs that work well on Minigrid including NovelD (Zhang et al., 2021), RIDE (Raileanu & Rocktäschel, 2020), and RND (Burda et al., 2018). For a fair comparison, we follow the same basic RL algorithm and network architectures used in the official codebase of NovelD and only change the intrinsic rewards r^{int} for all the methods. We also compare our method with EC (Savinov et al., 2018) because of the similarity of the high-level idea between the two methods. However, the original paper of EC does not include experiments on Minigrid. Therefore, we implement our own version to adapt to Minigrid. We follow their implementation suggestions in the paper and tune the hyper-parameters such as novelty threshold by grid search.

Dynamics Model Training For each experiment on Minigrid, we first run a random policy for $1e5$ steps to collect data and use them to train a forward dynamics model as the world model. Among the pairs collected, there are about $5e4$ different transition pairs. During our experiments, we observe that fine-tuning the pre-trained dynamics model during policy training has no significant influence on the performance. Similar to (Parisi et al., 2021), we use the 360° panoramic views as the input to predict the future observations. This is a rotation-invariant representation of the observed state. We consider this still a fair comparison with the previous state-of-the-arts because both NovelD and RIDE rely on using the state information instead of observations for the episodic count calculation.

Due to the limited field of view of the agent, we only forward the learned dynamics by $k = 1$ step when predicting. We predict the next observations produced by all 7 discrete actions in the Minigrid tasks including turn left, turn right, forward, toggle, pick up, drop, and done. We directly apply the default Python hashing function to hash the observations and predicted future observations. We do not expect the hashing function to mitigate the prediction error on Minigrid, but only use it to reduce the dimension of observations and predictions.

Training Performance on Minigrid Figure 5 shows the learning curves of GoBI and state-of-the-art exploration baselines NovelD, RIDE, RND, and EC on 12 most challenging Minigrid navigation tasks, including MultiRoom, KeyCorridor, and ObstructedMaze. Our curves are shifted towards right by the number of random exploration environment steps used to train the world model. In all 12 environments, GoBI significantly outperforms previous methods in terms of sample efficiency. For in-

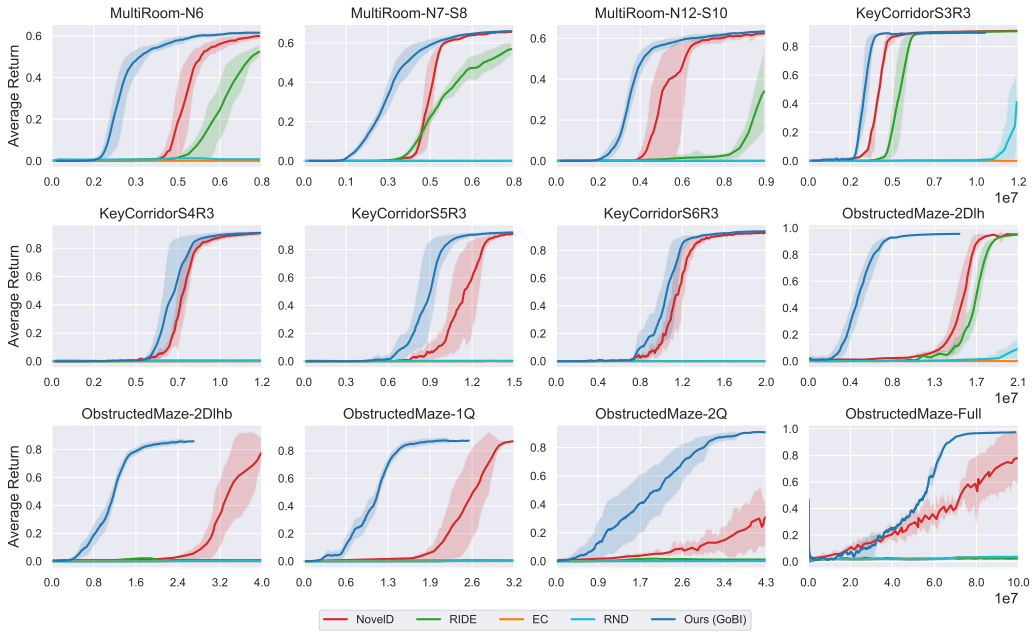


Figure 5. Training performance of GoBI and the baselines on 12 MiniGrid environments. The x-axis shows the number of environment steps. We shift the training curves towards right by the number of environment steps we use to pre-train the dynamics model, i.e. 1e5 time steps. Results are averaged across 4 seeds.

stance, on ObstructedMaze-2Dlhb, GoBI is about three times more sample efficient than NovelD. On the hardest ObstructedMaze-Full environment, GoBI achieves near-optimal performance within 70M steps. Lastly, although we try to tune the hyper-parameters of EC, our implementation of EC still does not learn well on the Minigrid environments.

Qualitative Results To clearly present the exploration behavior learned by GoBI, we show the visitation heatmaps of GoBI and NovelD on a KeyCorridorS5R3 environment in Figure 4. Not only does our method converge to an optimal policy faster, the exploration behaviour is very different from NovelD. GoBI quickly learns not to visit easily predictable states like an empty room, making it more efficient to explore interesting parts of the environment, for example, the room with a key in it.

3.2. Experiments on Control Tasks

We further test GoBI on DeepMind Control Suite, which are a set of image-based continuous control tasks. These tasks are more challenging than Minigrid because of its high-dimensional observations and stochastic transitions. Notice that these environments are not procedurally-generated. The experiments in this section are to show the generality of our method by experimentally showing that GoBI extends well to sparse-reward tasks with continuous action space and high-dimensional observation space.

Dynamics Model Training We follow the world model structure in Dreamer (Hafner et al., 2019) and directly apply their encoder, transition model, and observation model to predict future observations. However, compared to Minigrid, it requires way more data to train a decent dynamics model on DeepMind Control to generate visually-reasonable predictions. Therefore, different from the experiments on Minigrid, we do not pre-train the dynamics models. Instead we train the dynamics model together with the policy as is shown in Algorithm 1. We find that the number of sampled random actions $n = 5$ works well across all 4 environments. For the number of forward prediction steps k, we set it to be 3 for Pendulum Swingup and 1 for the other 3 environments.

For the hashing function, we find that a simple SimHash as is suggested in (Tang et al., 2017) works well in capturing the similarities between similar observations. We use SimHash to hash the image observations to 50 bits.

Training Performance on DeepMind Control We compare with the state-of-the-art intrinsic motivation on DeepMind Control tasks - RE3 (Seo et al., 2021), which applies a k-nearest neighbor entropy estimator in the low-dimensional representation space of a randomly initialized encoder to maximize state entropy. RE3 is also what we use for the lifelong intrinsic reward part r^{lifelong} of GoBI in Eq 1. Another two intrinsic reward baselines we consider are ICM (Pathak et al., 2017) and RND (Burda et al., 2018). For a fair comparison, all the experiments use the same basic RL algorithm

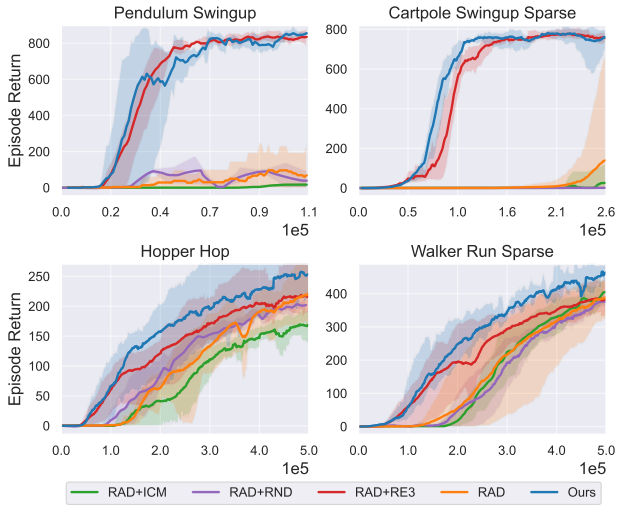


Figure 6. Training curves of GoBI and the baselines on DeepMind Control Suite. The curves are averaged across 5 seeds.

RAD (Laskin et al., 2020). The results are shown in Figure 6. The additional episodic-level intrinsic reward term improves the sample efficiency a lot compared to only using lifelong intrinsic reward, especially on Hopper Hop and Walker Run Sparse.

3.3. Ablation Study

GoBI Variations In this section, we analyze how each component of our intrinsic reward contributes to the final performance. We ablate each component of GoBI and run experiments on Minigrid environments with the following:

- R1: only episodic intrinsic reward $m_{t+1} - m_t$
- R2: indicator of whether new states are added to the episodic buffer $(\mathbb{1}\{m_{t+1} - m_t > 0\})/\sqrt{N(o_{t+1})}$
- R3: only lifelong intrinsic reward $1/\sqrt{N(o_{t+1})}$

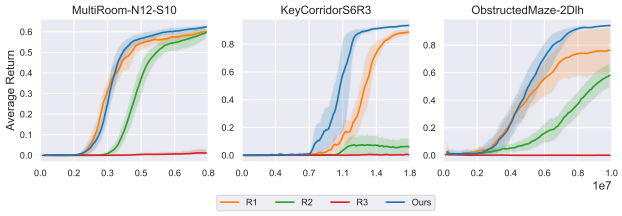


Figure 7. Training performance comparison among GoBI, R1, R2, and R3 on 3 Minigrid environments.

Training performance of GoBI as well as R1, R2, and R3 are shown in Figure 7. Although R1 works on MultiRoom, it suffers on Obstructed Maze and large KeyCorridor environments. The underlying reason may be that in Key Corridor and Obstructed Maze the room structures change less across episodes than MultiRoom (all generated rooms are squares with fixed sizes), therefore the COUNT-based rewards contribute more in such environments than in MultiRoom. At the same time, R2 performs way worse than GoBI. Agents trained with R2 prefer actions that only increase the size of the episodic buffer a bit therefore getting positive score more often. We provide an illustrative example in Appendix E to explain why R2 does not work well compared to GoBI. Using only lifelong intrinsic reward R3 performs the worst and struggles to learn efficiently on large Multiroom, Key Corridor, and Obstructed Maze environments.

Real Dynamics vs Learned Dynamics A learned dynamics model is generally not perfect, especially for partially-observable environments like Minigrid. In many cases the predictions can never be accurate. For example, when the agent first opens the door of a new room, usually it will not accurately predict everything behind the door. Figure 8 shows the training curves between using the real dynamics model vs a learned dynamics model. Not surprisingly, with the same intrinsic reward function, using the real dynamics converges faster to a near-optimal policy. However, even with imperfect dynamics model, our method still greatly surpasses previous state-of-the-arts.

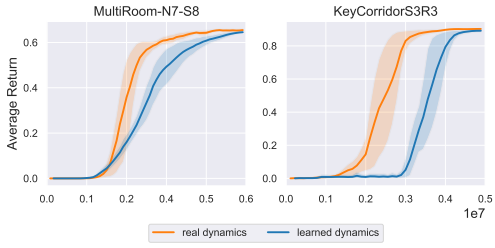


Figure 8. Comparison between using the real dynamics model of the environments vs using a learned one on Minigrid environments. In both MultiRoom and KeyCorridor, using a real dynamics model to derive intrinsic reward makes the policy converge faster, especially on KeyCorridor.

Multi-Step Predictions Figure 9 shows the learning performance of GoBI on Minigrid with a varying choices of the number of future steps to do predictions $k = 1, 2, 3$. For $k > 1$, our dynamics model outputs $pano_{t+1}$ instead of o_{t+1} and we hash and store the observations from panoramas in each future time step. With a real forward dynamics model, a larger k generally accelerates exploration more, because it prioritizes actions that lead to the states that are reachable to more states in the long run. However, due to the limited field of view of the agent and the model inaccuracy, this is not the case if we use a learned model. Forwarding 2 steps is still faster than only 1 step, but more steps than that does not really make exploration faster.

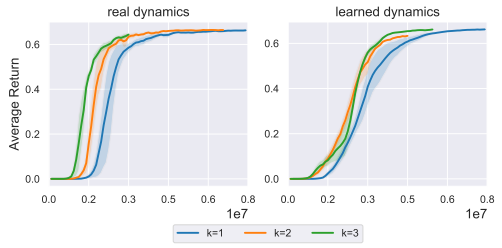


Figure 9. We make forward predictions for different number of future steps k using both the real dynamics and the learned dynamics model. The plots above show the training performance on Minigrid MultiRoom-N7-S8.

4. Related Work

4.1. Exploration in Reinforcement Learning

Efficient exploration in reinforcement learning, especially for sparse-reward reinforcement learning problems is challenging. A natural and popular solution is to design some metric to evaluate state novelty and assign high intrinsic reward to novel states. For example, COUNT-based intrinsic reward (Strehl & Littman, 2008; Kolter & Ng, 2009; Tang et al., 2017) and curiosity-based intrinsic motivation (Stadie et al., 2015; Pathak et al., 2017; Burda et al., 2018). Another popular way is to do state space entropy maximization (Hazan et al., 2019; Lee et al., 2019). Recently, nearest neighbor entropy estimation methods (Yarats et al., 2021a; Liu & Abbeel, 2021) have shown great performance improvements in challenging visual domains. Our method is compatible with all these successful exploration intrinsic reward designs by using them as r^{lifelong}, but we additionally encourage the episodic-level reachable space expansion to achieve large state space coverage within a single episode.

4.2. Episodic Memory

Deriving useful information from episodic buffer have shown great success in improving the training sample efficiency in RL on navigation, control, and Atari games. Episodic memory buffers are applied to mimic hippocampal episodic control and rapidly assimilate recent experience (Blundell et al., 2016; Pritzel et al., 2017). As is mentioned in the previous sections, (Savinov et al., 2018) keeps an episodic buffer to store observations and introduce an episodic curiosity module to determine if a new observation is reachable from previous observations or not. RAPID (Zha et al., 2021) proposes a novel way to do behaviour cloning on episodes with high episodic coverage. NGU (Badia et al., 2020) combines an episodic novelty module and a lifelong novelty module to generate intrinsic rewards. However, in NGU, the episodic novelty is a measurement of difference between the current observations from the previous observations, while ours focus on how much the reachable space is expanded from the new state. RIDE (Raileanu & Rocktäschel, 2020) and NovelD (Zhang et al., 2021) both count the episodic state visitations, while we claim that apart from visited states, we should also consider states that can be predicted from short-term episodic memory.

4.3. Learning World Models with Forward Dynamics

Learning dynamics function from a set of observed data is a widely-studied topic in reinforcement learning, especially due to the rapid growth of model-based reinforcement learning (Wang et al., 2019). Existing work show that an agent's world model is implicitly a forward model that predict future states (Ha & Schmidhuber, 2018a; Freeman et al., 2019). Recently, people have proposed latent dynamics models that work well on high-dimensional inputs (Okada et al., 2020). These latent dynamics models encode image observations and predict future states in the latent space (Ha & Schmidhuber, 2018b; Hafner et al., 2019; 2023), outputting realistic future observations on visually complex domains including DeepMind Control Suite (Tunyasuvunakool et al., 2020), VizDoom (Kempka et al., 2016), Atari Games, and DeepMind Lab (Beattie et al., 2016). The learned dynamics models can be used to guide exploration by prediction error (Stadie et al., 2015; Pathak et al., 2017; Burda et al., 2018), surprise (Achiam & Sastry, 2017), or information gain by variance of model ensemble means (Sekar et al., 2020). Our method differ from the previous methods by directly generating and hashing the predicted states and add them to an episodic reachable state buffer. With the advanced world model structures, our method can be extended to diverse domains with complex observations.

5. Discussions and Future Work

This paper shows an effective way to combine learned world models with episodic memory to intrinsically guide efficient exploration. Our method achieves state-of-the-art performance on procedurally-generated hard exploration tasks and also works well on singleton continuous control domains. However, it still has certain limitations. First of all, the dynamics model we use for the Minigrid experiments is deterministic, making it possible to generate less accurate predictions and making the performance of our method worse than using the real dynamics. A possible way to make improvement on this is to make the prediction model generative and sample possible future states. Secondly, for the control tasks with complex visual inputs, we hash the images with static hashing to make them discrete hash codes. However, to better capture the semantic similarities between the image observations, it would be beneficial to learn hash functions, for example, by using an autoencoder (AE) to learn meaningful hash codes (Tang et al., 2017). We leave these investigations as future work.

6. Conclusion

In this work, we introduce Go Beyond Imagination- GoBI, a novel episodic intrinsic reward design that encourages efficient episodic-level exploration by expanding reachable space. While most previous episodic intrinsic rewards use a naive episodic state count or state visitation coverage, our method exploits learned world models to predict reachable states and motivates the agent to seek for the states with more unexplored neighbors. Combined with lifelong intrinsic rewards, our method shows great training time sample efficiency improvement on hard procedurally-generated environments. At the same time, it can be extended to guide exploration on continuous control tasks with visual inputs, both indicating a promising future in this direction.

7. Acknowledgments

This work was supported in part by grants from LG AI Research, NSF IIS 1453651, and NSF FW-HTF-R 2128623.

References

Achiam, J. and Sastry, S. Surprise-based intrinsic motivation for deep reinforcement learning. *arXiv preprint arXiv:1703.01732*, 2017.

Andersen, P., Morris, R., Amaral, D., Bliss, T., and O'Keefe, J. *The hippocampus book*. Oxford university press, 2006.

Badia, A. P., Sprechmann, P., Vitvitskyi, A., Guo, D., Piot, B., Kapturowski, S., Tieleman, O., Arjovsky, M., Pritzel, A., Bolt, A., et al. Never give up: Learning directed exploration strategies. *arXiv preprint arXiv:2002.06038*, 2020.

Beattie, C., Leibo, J. Z., Teplyashin, D., Ward, T., Wainwright, M., Küttler, H., Lefrancq, A., Green, S., Valdés, V., Sadik, A., et al. Deepmind lab. *arXiv preprint arXiv:1612.03801*, 2016.

Blundell, C., Uria, B., Pritzel, A., Li, Y., Ruderman, A., Leibo, J. Z., Rae, J., Wierstra, D., and Hassabis, D. Model-free episodic control. *arXiv preprint arXiv:1606.04460*, 2016.

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

Chevalier-Boisvert, M., Willems, L., and Pal, S. Minimalistic gridworld environment for openai gym. https://github.com/maximecb/gym-minigrid, 2018a.

Chevalier-Boisvert, M., Willems, L., and Pal, S. Minimalistic gridworld environment for gymnasium, 2018b. URL https://github.com/Farama-Foundation/Minigrid.

Cobbe, K., Klimov, O., Hesse, C., Kim, T., and Schulman, J. Quantifying generalization in reinforcement learning. In *International Conference on Machine Learning*, pp. 1282–1289. PMLR, 2019.

Cobbe, K., Hesse, C., Hilton, J., and Schulman, J. Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*, pp. 2048–2056. PMLR, 2020.

Eichenbaum, H. The role of the hippocampus in navigation is memory. *Journal of neurophysiology*, 117(4):1785–1796, 2017.

Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International conference on machine learning*, pp. 1407–1416. PMLR, 2018.

Flet-Berliac, Y., Ferret, J., Pietquin, O., Preux, P., and Geist, M. Adversarially guided actor-critic. *arXiv preprint arXiv:2102.04376*, 2021.

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

Freeman, D., Ha, D., and Metz, L. Learning to predict without looking ahead: World models without forward prediction. *Advances in Neural Information Processing Systems*, 32, 2019.

Ha, D. and Schmidhuber, J. Recurrent world models facilitate policy evolution. *Advances in neural information processing systems*, 31, 2018a.

Ha, D. and Schmidhuber, J. World models. *arXiv preprint arXiv:1803.10122*, 2018b.

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

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

Hazan, E., Kakade, S., Singh, K., and Van Soest, A. Provably efficient maximum entropy exploration. In *International Conference on Machine Learning*, pp. 2681–2691. PMLR, 2019.

Kempka, M., Wydmuch, M., Runc, G., Toczek, J., and Jaśkowski, W. Vizdoom: A doom-based ai research platform for visual reinforcement learning. In *2016 IEEE conference on computational intelligence and games (CIG)*, pp. 1–8. IEEE, 2016.

Kirk, R., Zhang, A., Grefenstette, E., and Rocktäschel, T. A survey of generalisation in deep reinforcement learning. *arXiv preprint arXiv:2111.09794*, 2021.

Kolter, J. Z. and Ng, A. Y. Near-bayesian exploration in polynomial time. In *Proceedings of the 26th annual international conference on machine learning*, pp. 513–520, 2009.

Laskin, M., Lee, K., Stooke, A., Pinto, L., Abbeel, P., and Srinivas, A. Reinforcement learning with augmented data. *Advances in neural information processing systems*, 33: 19884–19895, 2020.

Lee, L., Eysenbach, B., Parisotto, E., Xing, E., Levine, S., and Salakhutdinov, R. Efficient exploration via state marginal matching. *arXiv preprint arXiv:1906.05274*, 2019.

Liu, H. and Abbeel, P. Behavior from the void: Unsupervised active pre-training. *Advances in Neural Information Processing Systems*, 34:18459–18473, 2021.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pp. 1928–1937. PMLR, 2016.

Okada, M., Kosaka, N., and Taniguchi, T. Planet of the bayesians: Reconsidering and improving deep planning network by incorporating bayesian inference. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5611–5618. IEEE, 2020.

Parisi, S., Dean, V., Pathak, D., and Gupta, A. Interesting object, curious agent: Learning task-agnostic exploration. *Advances in Neural Information Processing Systems*, 34, 2021.

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

Pritzel, A., Uria, B., Srinivasan, S., Badia, A. P., Vinyals, O., Hassabis, D., Wierstra, D., and Blundell, C. Neural episodic control. In *International Conference on Machine Learning*, pp. 2827–2836. PMLR, 2017.

Raileanu, R. and Rocktäschel, T. Ride: Rewarding impact-driven exploration for procedurally-generated environments. *arXiv preprint arXiv:2002.12292*, 2020.

Riedmiller, M., Hafner, R., Lampe, T., Neunert, M., Degrave, J., Wiele, T., Mnih, V., Heess, N., and Springenberg, J. T. Learning by playing solving sparse reward tasks from scratch. In *International conference on machine learning*, pp. 4344–4353. PMLR, 2018.

Savinov, N., Raichuk, A., Marinier, R., Vincent, D., Pollefeys, M., Lillicrap, T., and Gelly, S. Episodic curiosity through reachability. *arXiv preprint arXiv:1810.02274*, 2018.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Sekar, R., Rybkin, O., Daniilidis, K., Abbeel, P., Hafner, D., and Pathak, D. Planning to explore via self-supervised world models. In *International Conference on Machine Learning*, pp. 8583–8592. PMLR, 2020.

Seo, Y., Chen, L., Shin, J., Lee, H., Abbeel, P., and Lee, K. State entropy maximization with random encoders for efficient exploration. In Meila, M. and Zhang, T. (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 9443–9454. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/seo21a.html.

Stadie, B. C., Levine, S., and Abbeel, P. Incentivizing exploration in reinforcement learning with deep predictive models. *arXiv preprint arXiv:1507.00814*, 2015.

Strehl, A. L. and Littman, M. L. An analysis of model-based interval estimation for markov decision processes. *Journal of Computer and System Sciences*, 74(8):1309–1331, 2008.

Tang, H., Houthooft, R., Foote, D., Stooke, A., Xi Chen, O., Duan, Y., Schulman, J., DeTurck, F., and Abbeel, P. # exploration: A study of count-based exploration for deep reinforcement learning. *Advances in neural information processing systems*, 30, 2017.

Tunyasuvunakool, S., Muldal, A., Doron, Y., Liu, S., Bohez, S., Merel, J., Erez, T., Lillicrap, T., Heess, N., and Tassa, Y. dm_control: Software and tasks for continuous control. *Software Impacts*, 6:100022, 2020. ISSN 2665-9638. doi: https://doi.org/10.1016/j.simpa.2020.100022. URL https://www.sciencedirect.com/science/article/pii/S2665963820300099.

Wang, T., Bao, X., Clavera, I., Hoang, J., Wen, Y., Langlois, E., Zhang, S., Zhang, G., Abbeel, P., and Ba, J. Benchmarking model-based reinforcement learning. *arXiv preprint arXiv:1907.02057*, 2019.

Yarats, D., Fergus, R., Lazaric, A., and Pinto, L. Reinforcement learning with prototypical representations. In *International Conference on Machine Learning*, pp. 11920–11931. PMLR, 2021a.

Yarats, D., Zhang, A., Kostrikov, I., Amos, B., Pineau, J., and Fergus, R. Improving sample efficiency in model-free reinforcement learning from images. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 10674–10681, 2021b.

Zha, D., Ma, W., Yuan, L., Hu, X., and Liu, J. Rank the episodes: A simple approach for exploration in procedurally-generated environments. *arXiv preprint arXiv:2101.08152*, 2021.

Zhang, T., Xu, H., Wang, X., Wu, Y., Keutzer, K., Gonzalez, J. E., and Tian, Y. Noveld: A simple yet effective exploration criterion. *Advances in Neural Information Processing Systems*, 34, 2021.

A. Implementation Details

A.1. Experiments on Minigrid

Baselines Implementations of GoBI, NovelD (Zhang et al., 2021), RIDE (Raileanu & Rocktäschel, 2020), RND (Burda et al., 2018), and EC (Savinov et al., 2018) are built on the official codebase of NovelD. For fair comparisons, only the intrinsic reward r^{int} differs among the methods and they all use the same base algorithm IMPALA (Espeholt et al., 2018). At the same time, all the experiments are run with the same compute resource with Nvidia TITAN X GPU and 40 CPUs. For NovelD, we rerun their official code to get the results of Minigrid MultiRoom and KeyCorridor. For the experiments on ObstructedMaze, we did not find the proper hyper-parameters to fully reproduce their results. Therefore, we directly take the results reported in their paper. For RIDE and RND, we run the code in the official codebase of NovelD. For EC (Savinov et al., 2018), their original paper does not include experiments on Minigrid environments. Therefore, we implement our own version and tune the hyper-parameters with grid search. The intrinsic reward functions of GoBI and the baselines are listed below:

- GoBI: $(m_{t+1} - m_t)/\sqrt{N(o_{t+1})}$, where m_t is the size of the episodic buffer $\mathcal{M}$ and $N(o_{t+1})$ is the lifelong count of the observation o_{t+1} starting from the beginning of training.
- RND: $\|\phi(o_{t+1}) - \hat{\phi}(o_{t+1})\|^2$, which is the difference between a fixed random network $\hat{\phi}$ and a trained state embedding network ϕ. Here, ϕ is trained to minimize the same error.
- NovelD: $\max[novelty(o_{t+1}) - \alpha \cdot novelty(o_t), 0] * \mathbb{1}\{N^{epi}(s_{t+1} = 1)\}$. They apply RND to measure the novelty of o_t, i.e., $novelty(o_t) = \|\phi(o_t) - \hat{\phi}(o_t)\|^2$. $N^{epi}(s_{t+1} = 1)$ checks if the agent visits state s_{t+1} for the first time in an episode. Notice that they use the full environment information, i.e. everything in the grid world instead of only the 7×7 partially-observable view. Therefore N^{epi} counts s_{t+1} instead of o_{t+1}.
- RIDE: $\|\phi(o_t) - \phi(o_{t+1})\|^2/\sqrt{N^{epi}(s_{t+1})}$, where ϕ is the state embedding network trained to minimize the prediction error of an inverse and a forward dynamics. N^{epi} indicates the episodic counts. Same as NovelD, in RIDE, they also use the state information s_{t+1} for episodic count.
- EC: $\beta - C(\mathcal{M}, o_{t+1})$, where $C(\mathcal{M}, o_{t+1})$ is the 90-th percentile similarity scores between o_{t+1} and all the observations in the episodic buffer $\mathcal{M}$. The similarity scores are calculated using a pre-trained episodic curiosity module. β is a hyper-parameter.

Policy and Value Function Training For fair comparisons, the policy network and value function network are the same for all approaches. The input observations of dimension $7 \times 7 \times 3$ are put into a shared feature extraction network, which includes three convolutional layers of kernel size$= 3 \times 3$, padding$= 1$, channel$=32, 128, 512$, and stride$= 1, 2, 2$ respectively with ELU activation. The features are then flattened and put through 2 linear layers with 1024 units and ReLU activation, and an LSTM layer with 1024 units. This shared feature is passed separately to 2 fully-connected layers with 1024 units to output action distribution and value estimation.

Dynamics model For our implementation of the dynamics model, our input is the panorama of the current step. To get the panorama, we let the agent rotate for 3 times and concatenate the 4 observations to get inputs of size $28 \times 7 \times 3$. It is then passed to a feature extraction module that has the same structure as our policy and value function networks, except that the input to the first linear layer is 4×1024. We then concatenate it with actions and put it through a decoder with 2 linear layers of sizes 256 and 512, and reshape back to $7 \times 7 \times 3$ to get a predicted observation. We pretrain the dynamics model using 1e5 $(pano_t, a_t, o_{t+1})$ pairs collected by a random policy. RIDE also requires training dynamics models for the state embedding network ϕ. The input of their dynamics model is the state embedding and action. The forward model contains two fully-connected layers with 256 and 128 units activated by ReLU. The inverse dynamics model contains two fully-connected layers with 256 units and a ReLU activation function. Its input is the state embeddings of two consecutive steps.

Hash Functions We directly apply the default Python hashing function to hash the $7 \times 7 \times 3$ observations and predicted future observations before adding them to the episodic buffer.

State embedding NovelD, RIDE, and RND all require training a state embedding network ϕ. The input is the observation in MiniGrid with dimension $7 \times 7 \times 3$. It contains three convolutional layers with kernel size$= 3 \times 3$, padding $= 1$, stride $= 1, 2, 2$, number of channels $= 32, 128, 512$ respectively. The activation function is ELU. Following the convolutional layers are two linear layers of 2048 and 1024 units with ReLU activation.

Visitation Count For GoBI, $N(o)$ stores the flattened $7 \times 7 \times 3$ observations of each step. And for NovelD and RIDE, they count the full states at episodic level, whose shape varies from environment to environment. For example, the shape is $25 \times 25 \times 3$ for MultiRoom environments.

Hyper-parameters Table 1 shows the values of hyper-parameters shared across different methods.

Parameter name	Value
Batch Size	32
Optimizer	RMSProp
Learning Rate	0.0001
LSTM Steps	100
Discount Factor γ	0.99
Weight of Policy Entropy Loss	0.0005
Weight of Value Function Loss	0.5

Table 1. Hyper-parameters for experiments on Minigrid. These hyper-parameters are shared across all the methods

For all of our experiments using GoBI on Minigrid, we set the intrinsic reward coefficient $\lambda = 0.01$ and $k = 1$, which means only forwarding the dynamics model by 1 step. At the same time, as the action space is small and discrete, instead of randomly sampling some actions, we directly predict the future observations using all 7 possible actions. We list the hyper-parameter choices of intrinsic decay factor in Table 2. The value of ρ is chosen to make the intrinsic reward large at the beginning of training and near-zero at the end of the training.

Parameters	Value
Forward Step k	1
Intrinsic Decay ρ	6e-7 for MR-N7S8; 8e-7 for MR-N12S10, MR-N6; 1.5e-6 for KC-S3R3; 5e-7 for KC-S4R3, KC-S5R3 3e-7 for KC-S6R3, OM-2Dlh 2e-7 for OM-1Q, OM-2Dlhb, OM-2Q 5e-8 for OM-Full
$\hat{f}_\phi$ Optimizer	Adam
$\hat{f}_\phi$ Learning Rate	5e-4

Table 2. The hyper-parameters of GoBI for experiments on Minigrid.

For NovelD, we set $\lambda = 0.05$ for all the environments as is suggested in their official codebase. For RIDE, we use $\lambda = 0.1$ on KeyCorridor-S3R3 and $\lambda = 0.5$ on all other environments. For RND, we set $\lambda = 0.1$ on all the environments. For EC, we make $\lambda = 0.01$ so that the initial average intrinsic reward of EC is similar to ours.

A.2. Experiments on Deepmind Control Suites

Baselines Implementations of GoBI, RE3 (Seo et al., 2021), ICM (Pathak et al., 2017), and RND (Burda et al., 2018) are built on the official codebase of RE3. All the experiments apply the same base reinforcement learning algorithm RAD (Laskin et al., 2020). For RE3, we rerun their official code to get the results on all four environments. For ICM and RND, we follow the implementation details listed in RE3 to implement them to be compatible with DeepMind Control tasks. For a fair comparison, only the intrinsic reward design differs among the methods. The intrinsic reward functions of GoBI and the baselines are listed below:

- GoBI: $(m_{t+1} - m_t) \times \log(||y_i - y_i^{k-NN}||_2 + 1)$, where m_t is the size of the episodic buffer $\mathcal{M}$. The latter part is the RE3 intrinsic reward which we introduce below.
- RE3: $\log(||y_i - y_i^{k-NN}||_2 + 1)$, where $y_i = f_\theta(s_i)$ is a fixed representation outputs from a randomly initialized encoder and y_i^{k-NN} is a set of k-nearest neighbors of y_i among all the collected y's from the beginning of training.
- ICM: $\frac{\eta}{2}||\hat{\phi}(o_{t+1}) - \phi(o_{t+1})||_2^2$, where η is a scaling factor. $\phi(o)$ is a feature vector that is jointly optimized with a forward prediction model and an inverse dynamics model and $\hat{\phi}(o)$ predicts the feature encoding at time step $t+1$.
- RND: $\|\phi(o_{t+1}) - \hat{\phi}(o_{t+1})\|^2$, which is the difference between a fixed random network $\hat{\phi}$ and a trained state embedding network ϕ. Here, ϕ is trained to minimize $\|\phi(o_{t+1}) - \hat{\phi}(o_{t+1})\|^2$.

Architecture The observation size of all the environments is $84 \times 84 \times 3$. The encoder architecture follows the same one as in (Yarats et al., 2021b), which contains 4 convolutional layers of 3×3 kernels, channel=32, and stride=$2, 1, 1, 1$ with ReLU activations. The output is then passed to a fully-connected layer and normalized by LayerNorm.

Dynamics Model For the forward dynamics model that we use to generate future predictions, we apply the same world model structure as in Dreamer (Hafner et al., 2019). The input size of Dreamer is $64 \times 64 \times 3$. We down-sample the input observations to 64×64 instead of tuning the world model layers. We train the dynamics model together with the RL policy in an online manner instead of pre-train it because it takes many episodes for the predictions to be visually reasonable. Therefore it does not add extra effort to determine how many data should we collect to pre-train the dynamics model.

Image hashing As the observations are images in high-dimensional space and the predictions are usually not accurate, we hash the images to lower dimension to avoid taking too much space and to collapse similar observations and

predictions. Following (Tang et al., 2017), we use the simple SimHash function to map the images to 50 bits. More specifically, we project the flattened images to a random initialized vector and use the signs of output vector values as the hash code.

Hyper-parameters Table 3 shows the values of hyper-parameters shared across different methods.

Parameter name	Value
Augmentation	Crop
Observation Size	(84, 84)
Action Repeat	2
Replay Buffer Size	100000
Initial Random Exploration Steps	1000
Frame Stack	3
Actor Learning Rate	0.0002
Critic Learning Rate	0.0002
Batch Size	512
# Nearest Neighbors	3
Critic Target Update Freq	2

Table 3. Hyper-parameters for experiments on DeepMind Control Suites. These hyper-parameters are shared across all the methods

For the intrinsic reward coefficients, we follow the best choices reported in RE3. For GoBI, we apply the same intrinsic reward coefficient λ and intrinsic reward decay ρ as the ones in RE3 for fair comparison. The intrinsic rewards that are specific to our method is shown in Table 4. For the number of random actions n, we perform hyper-parameter search over $\{3, 5, 10, 20\}$ and find that $n = 5$ perform well across all the tasks. For the number of forward steps k, we perform hyper-parameter search over $\{1, 2, 3, 5\}$ and report the ones with the best results.

Parameter name	Value
# Forward Step k	3 for pendulum-swingup; 1 for others
# Random Actions n	5

Table 4. Hyper-parameters for experiments on DeepMind Control Suites. These hyper-parameters are specific to our method.

B. Hyper-Parameters

B.1. Intrinsic reward decay

In Figure 10, we show the training performance with different intrinsic reward decay ρ. The choice of ρ is to balance between the relative importance of the extrinsic and intrinsic reward. If ρ is too large, for example when $\rho = 1\mathrm{e} - 6$, before the agent finds any goal, the intrinsic reward already decreases to very small. Therefore sometimes it is hard for the agent to learn anything useful, resulting in unsatisfactory performance. Meanwhile, if ρ is too small, for example when $\rho = 5\mathrm{e} - 7$, the intrinsic reward will be too large at later stage of training and make the agent focus less on the extrinsic reward. Therefore the policy may converge slower.

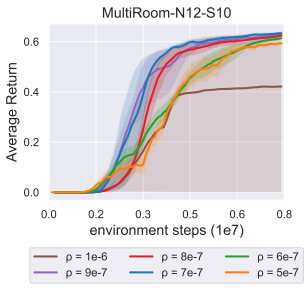


Figure 10. Training performance of GoBI on Minigrid Multiroom-N12-S10 with different intrinsic reward decay ρ.

An effective way to tune this hyper-parameter is to also record the number of (x, y) positions that the agent visits. If the visited area is large but the average return is low, we know that the intrinsic reward is too large and the agent ignores the extrinsic reward.

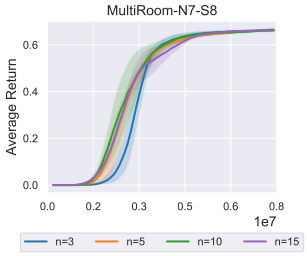


Figure 11. Training performance of GoBI on Minigrid Multiroom-N7-S8 with different n randomly sampled actions per step.

B.2. Number of randomly sampled actions

In the Minigrid experiments reported in Section 3, we do not randomly sample actions because Minigrid has a small discrete action space with only 7 actions. Therefore we directly predict future observations of all 7 actions. However, we also report the results with $n = 3, 5, 10, 15$ random actions in Figure 11. To summarize, $n = 5, 10, 15$ all have similar performance on Minigrid, while a larger n makes the wallclock training time longer. $n = 3$ is slightly slower at the early stage, but still outperforms the previous state-of-the-arts. Overall, $n = 5$ would be a good choice for the Minigrid environments.

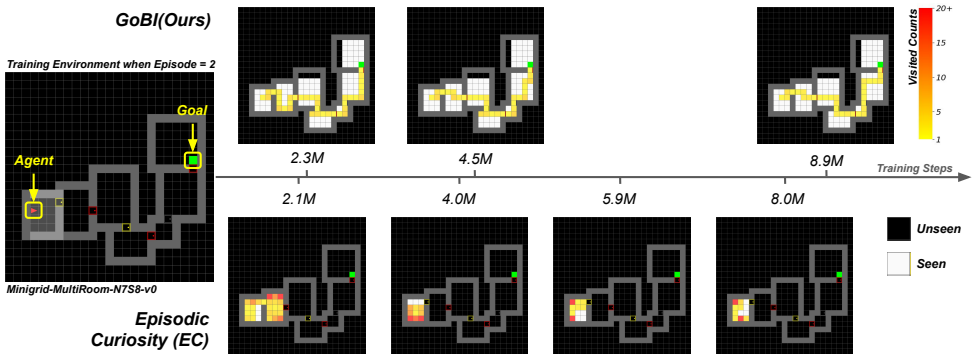


Figure 12. Heatmaps of example trajectories at different training steps in Minigrid-Multiroom-N7-S8 environment.

C. Exploration Behaviour Comparison between GoBI and EC

In Figure 12, we visualize the policy visitation heatmaps of GoBI and EC (Savinov et al., 2018) on a MultiRoom environment from Minigrid. Although EC fails to learn an optimal policy, we can still capture its preference from the heatmaps. An agent trained with EC prefers going to the corners of the room, which generally have lower similarity scores than the states in the middle of the room. However, if the similarity scores are not low enough for the states to be added to the episodic buffer, it will continue staying at the states to maximize its intrinsic reward. We tried to tune the similarity score threshold using grid search but still have not find a good hyper-parameter choice for it because the similarity score at different corners does not share a consistent value. Unlike EC, we can see from the figure that GoBI chooses not to visit the border of the room early on in training, as the information on the border are easily predictable from the information in the middle of the room.

D. Wallclock Training Time

Table 5 shows the wallclock time needed to train NovelD and our method for 10M Minigrid environment steps. GoBI requires about 2x the wallclock time needed to train NovelD for the same number of environment steps.

Algorithm	Wallclock Time (hours)
NovelD	$5.46(\pm 0.058)$
GoBI(ours)	$10.65(\pm 0.082)$

Table 5. Wallclock training time comparison in hours between NovelD and GoBI.

E. Ablation Study Illustration

In this section, we provide an illustrative example of why only considering whether new states are added to the episodic buffer or not, i.e., R2 in Section 3.3, works way worse than our method. The example is shown in Figure 14. If these 9 states are only a small part of the environment, we want a policy that explore this part as quickly as possible - mark all the states as reachable as quickly as possible. However, in order to maximize its step-wise intrinsic reward, an agent trained with R2 will go along the border to only add a few new states to the episodic buffer at a time, which wastes many unnecessary steps so is not beneficial for exploration.

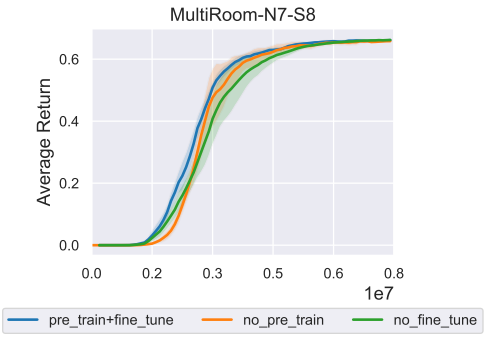


Figure 13. Training performance comparison on MultiRoom-N7-S8 environment among 1) use a fixed pre-trained dynamics model, 2) use a pre-trained dynamics model, and fine-tune it online, 3) no pre-training, directly train the dynamics model together with policy training.

F. Dynamics Training

In the experiment section 3, we report the results of applying a pre-trained forward dynamics for GoBI on Minigrid. However, the forward dynamics model can also be trained together with policy training. In Figure 13, we report the results of an ablation study on a Minigrid environment of 3 settings: 1) use a pre-trained dynamics model, and keep it fixed when training the policy, 2) pre-train a dynamics model, and fine-tune it when training the policy, 3) no pre-training, directly train the dynamics model in an online manner. In summary, all 3 versions work similarly, but due

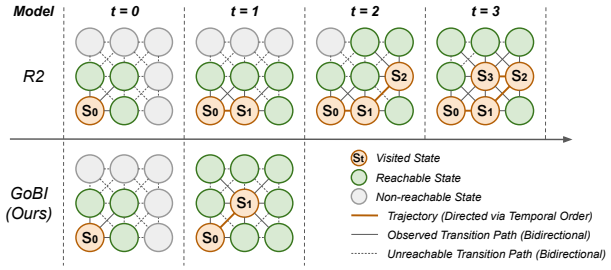


Figure 14. An illustrative example of why R2 does not work well to encourage efficient exploration. In this example, our goal is to include all the states into the episodic buffer quickly. GoBI can move directly to the center in one step for maximum intrinsic reward, while R2 may choose to take extra steps for exploration since it mainly focuses on whether the episodic buffer expands or not.

to the fact that training the dynamics model online will add extra wall clock training time, we use option 1 in our main experiments.