A HYPERPARAMETERS AND NEURAL NETWORK ARCHITECTURE

For the proposed RAPID, the common hyperparameters are summarized in Table 2. RAPID is based on the PPO implementation from OpenAI baselines[1]. The nstep is 128 for all MiniGrid environments and MuJoCo environments, and 512 for MiniWorld-Maze-S5. For MuJoCo environments, we report the results with only imitation loss since we find that the policy loss of PPO will harm the performance in the episodic reward setting. The learning rate is 10^{-4} for all MiniGrid environments and MiniWorld-Maze-S5, and 5×10^{-4} for MuJoCo environments. Since MiniWorld-Maze-S5 and MuJoCo environments have continuous state space, we set $w_2 = 0$. In practice, in MiniWorld-Maze-S5, we observe that counting the states will usually lead to memory error because it is not likely to encounter the same state again in MiniWorld-Maze-S5 (COUNT is excluded in comparison due to memory issues). Note that it is possible to use pseudo-counts (Bellemare et al., 2016; Ostrovski et al., 2017), which we will study in our future work. For MiniGrid-KeyCorridorS2R3-v0, MiniGrid-KeyCorridorS3R3-v0 and MiniWorld-Maze-S5, the update frequency of imitation learning is linearly annealed to 0 throughout the training process. For other environments, the imitation learning is performed after each episode. We keep all other hyperparameters of PPO as default and use the default CNN architecture provided in OpenAI baselines for MiniWorld-Maze-S5 and MLP with 64-64 for other environments. The PPO also uses the same hyperparameters. We summarize the state space of each environment and how the local and global scores are computed in Table 3.

Hyperparameter	**Value**
w_0	1
w_1	0.1
w_2	0.001
buffer Size	10000
batch Size	256
number of update steps	5
entropy coefficient	0.01
value function coefficient	0.5
γ	0.99
λ	0.95
clip range	0.2

Table 2: Common hyperparameters of the proposed RAPID (top) and the hyperparameters of PPO baseline (bottom).

Environment	**State Space**	**Local Score**	**Global Score**
8 MiniGrid Environments	$7 \times 7 \times 3$, discrete	Eq. 1	Eq. 3
MiniWorld Maze	$60 \times 80 \times 3$, continuous	Eq. 2	No
MuJoCo Walker2d	17, continuous	Eq. 2	No
MuJoCo Walker2d	11, continuous	Eq. 2	No
MuJoCo Walker2d	4, continuous	Eq. 2	No
MuJoCo Walker2d	8, continuous	Eq. 2	No

Table 3: The calculation of local and global scores for all the environments. Note that if the state space is continuous. We disable the global score since counting continuous states is meaningless. A possible solution is to use pseudo-counts (Bellemare et al., 2016; Ostrovski et al., 2017).

For RANDOM, we implement the two networks with 64-64 MLP. For CURIOSITY and RIDE, we use 64-64 MLP for state embedding model, forward dynamics model and inverse dynamics model, respectively. However, with extensive hyperparameters search, we are not able to reproduce the RIDE results in MiniGrid environments even with the exactly same neural architecture and hyperparameters as suggested in (Raileanu & Rocktäschel, 2020). Thus, we use the authors' implementation[2] (Raileanu & Rocktäschel, 2020), which is well-tuned on MiniGrid tasks. Our reported result

[1] `https://github.com/openai/baselines`

[2] `https://github.com/facebookresearch/impact-driven-exploration`

is on par with that reported in (Raileanu & Rocktäschel, 2020). For SIL and AMIGO, we use the implementations from the authors[3][4] with the recommended hyperparameters.

For the methods based on intrinsic rewards, i.e., RIDE,, AMIGO CURIOSITY, RANDOM, and COUNT, we search intrinsic reward coefficient from $\{0.1, 0.05, 0.01, 0.005, 0.001, 0.0005, 0.0001\}$. For RIDE, the intrinsic reward coefficient is 0.1 for MiniGrid-KeyCorridorS2R3-v0, MiniGrid-KeyCorridorS3R3-v0, MiniGrid-KeyCorridorS4R3-v0, MiniGrid-MultiRoom-N10S4-v0, MiniGrid-MultiRoom-N7S4-v0 and MiniWorld-Maze-S5, and 0.5 for MiniGrid-MultiRoom-N7S8-v0, MiniGrid-MultiRoom-N10S10-v0 and MiniGrid-MultiRoom-N12S10-v0. For AMIGO and CURIOSITY, the intrinsic reward coefficient is 0.1 for all the environments. For COUNT, the intrinsic reward coefficient is 0.005 for all the environments. For RANDOM, the intrinsic reward coefficient is 0.001 for all the environemnts. We also tune the entropy coefficient of RIDE from $\{0.001, 0.0005\}$, as suggested by the original paper. The entropy coefficient is 0.0005 for MiniGrid-KeyCorridorS2R3-v0, MiniGrid-KeyCorridorS3R3-v0, MiniGrid-KeyCorridorS4R3-v0, MiniGrid-MultiRoom-N10S4-v0, MiniGrid-MultiRoom-N7S4-v0 and MiniWorld-Maze-S5, and 0.001 for MiniGrid-MultiRoom-N7S8-v0, MiniGrid-MultiRoom-N10S10-v0 and MiniGrid-MultiRoom-N12S10-v0.

The w_0, w_1, and w_2 are selected as follows. We first run RAPID on MiniGrid-MultiRoom-N7S8-v0. We choose this environment because it is not too hard nor too easy, so we expect that it is representative among all the considered MiniGrid environments. We fix w_0 to be 1, and select w_1 and w_2 from $\{10^{-4}, 10^{-3}, 10^{-2}, 10^{-1}, 10^{0}, 10^{1}, 10^{2}, 10^{3}, 10^{4}\}$, that is, there are $9 \times 9 = 81$ combinations. Note that we can fix w_0 because only the relative ranking matters in the ranking buffer. The selected w_1 and w_2 are then directly used in other environments without tuning. We further conduct a sensitivity analysis on MiniGrid-MultiRoom-N7S8-v0 in Figure 8. We observe that RAPID has good performance in a very wide range of hyperparameter choices.

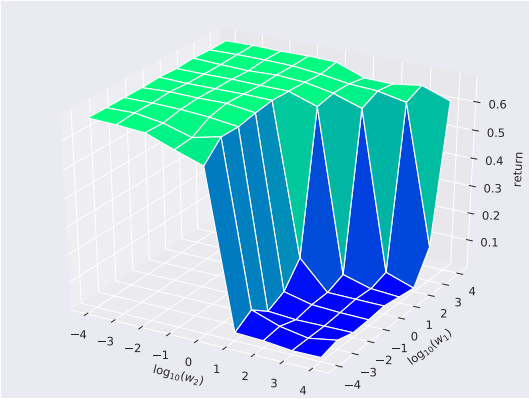


Figure 8: Sensitivity analysis of w_1 and w_2 (w_0 is fixed to 1) in MiniGrid-MultiRoom-N7S8-v0. All the experiments are run 3×10^6 timesteps. Note that 3×10^6 is more than enough for RAPID to converge in this environment (see Figure 4). The average results over 5 independent runs are plotted.

B Environments

B.1 MiniGrid Environments

MiniGrid[5] is a suite of light-weighted and fast gridworld environments with OpenAI gym interfaces. The environments are partially observable with a grid size of $N \times N$. Each tile in the gird can have at most one object, that is, 0 or 1 object. The possible objects are wall, floor, lava, door, key, ball, box and goal. Each object will have an associated color. The agent can pick up at most one object (ball or key). To open a locked door, the agent must use a key. Most of the environments in

[3] https://github.com/junhyukoh/self-imitation-learning
[4] https://github.com/facebookresearch/adversarially-motivated-intrinsic-goals
[5] https://github.com/maximecb/gym-minigrid

MiniGrid are procedurally-generated, i.e. a different grid will be sampled in each episode. Figure 9 shows the grids in four different episodes. The procedurally-generated nature makes training RL challenging because the RL agents need to learn the skills that can generalize to different grids. The environments can be easily modified so that we can create different levels of the environments. For example, we can modify the number of rooms and the size of each room to create hard-exploration or easy-exploration environments. The flexibility of MiniGrid environments enables us to conduct systematic comparison of algorithms under different difficulties.

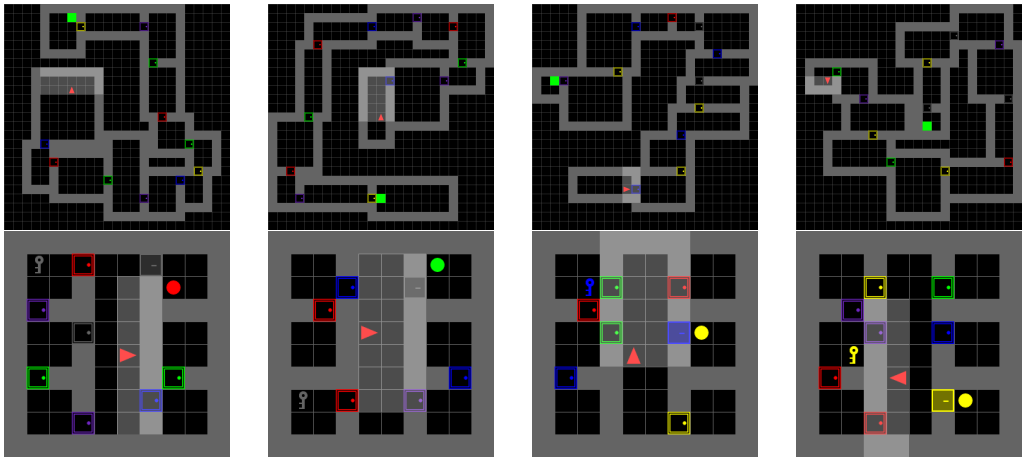


Figure 9: Rendering of MultiRoom-N12-S10 (top row) and KeyCorridor-S4-R3(bottom row) in 4 different episodes. The environments are procedually-generated, i.e., a different room is generated in a new episode.

The original observations of MiniGrid are dictionaries, which consist of a *image* field providing a partially observable view of the environment, and a *mission* field describing the goal with texts. In our experiments, we use `ImgObsWrapper` which only keeps the *image* field. The environment uses a compact encoding with 3 input values per visible grid cell. Note that the cells behind the wall or unopened doors are invisible.

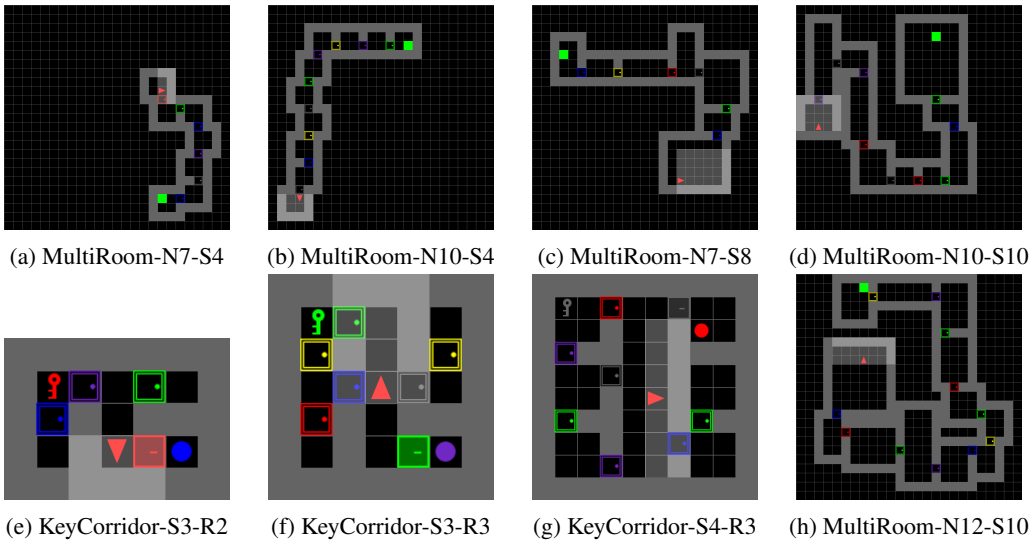
The possible actions in MiniGrid are (1) turn left, (2) turn right, (3) move forward, (4) pick up an object, (5) drop the carried object, (6) open doors/interact with objects, and (7) done. The agent will remain in the same state if the action is not legal. For example, if there is no object in front of the agent, the agent will remain in the same state when performing action (4) pick up an object.

We focus on MultiRoom-NX-SY and KeyCorridor-SX-RY, where X and Y are hyperparameters specifying the number of rooms or the room sizes in the environments. With larger X and Y, the environments will become more difficult to solve due to the sparse rewards. Figure 10 shows the environments (with different difficulties) in this work. In MultiRoom-NX-SY, the agent needs to navigate the green tile in the last room. If the agent successfully navigates the goal, the agent will receive a reward of 1 subtracting some penalties of the timesteps spent. In KeyCorridor-SX-RY, the agent needs to pick up the key, use the key to open the locked door, and pick up the ball. Similarly, the agent will receive a reward of 1 subtracting some penalties of the timesteps spent if it picks up the ball. Both environments are extremely difficult to solve using RL alone due to the sparse rewards.

B.2 MiniWorld Maze Environment

MiniWorld[6] is a minimalistic 3D interior environment simulator. Similar to Minigrid, we can easily create environments with different levels. We focus on MiniWorld-Maze, where the agent is asked to navigate the goal through a procedurally-generated maze. The agent has a first-person partially observable view of the environment. This environment is extremely difficult to solve due to sparse reward, long-horizon, and the procedurally-generated nature of the environments.

[6] `https://github.com/maximecb/gym-miniworld`



(a) MultiRoom-N7-S4 (b) MultiRoom-N10-S4 (c) MultiRoom-N7-S8 (d) MultiRoom-N10-S10

(e) KeyCorridor-S3-R2 (f) KeyCorridor-S3-R3 (g) KeyCorridor-S4-R3 (h) MultiRoom-N12-S10

Figure 10: Rendering of Minigird environments used in the work.

In this work, we focus on MiniWorld-Maze-S5, a variant of the MiniWorld Maze environment with 5×5 tiles. Figure 11 shows the top view of the mazes in four different episodes. In each episode, the agent and the goal are initialized in the top-left and bottom-right corners of the map, respectively. In this way, we ensure that the agent and the goal are far away enough so that the agent can not easily see the goal without exploration. A random maze will be generated in each episode. There are three possible actions: (1) move forward, (2) turn left, and (3) turn right. The agent will move $0.4 \times tile\_length$ if moving forward, where $tile\_length$ is the length of one side of a tile. The agent will rotate 22.5 degrees if turning right/left. The time budget of each episode is 600. The agent will not receive any positive reward if it can not navigate the goal under the time budget.

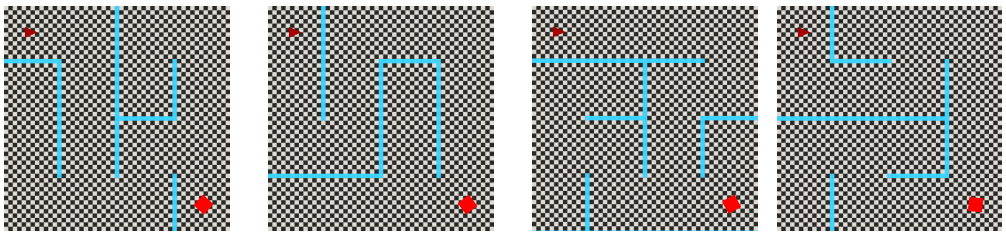
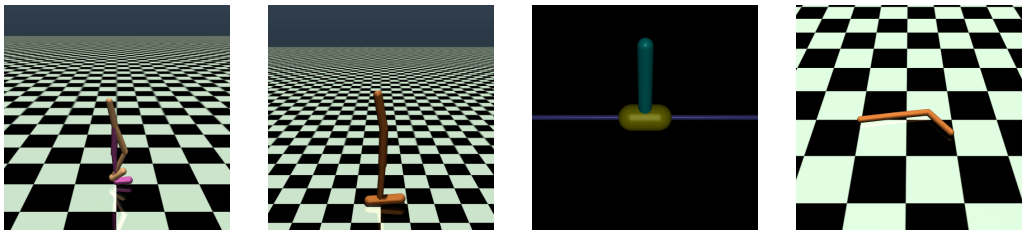


Figure 11: Example generated mazes in 4 diffident episodes.

B.3 Sparse MuJoCo Environments

Mujoco is a physical engine for continuous control (Todorov et al., 2012). Figure 12 shows the rendering of the four MuJoCo tasks used in our work. The original MuJoCo environments use dense rewards, i.e., a well-defined reward is given in each timestep. However, in many scenarios, well-defined dense rewards may be not available. We consider a variant of MuJoCo task using only episodic reward. Specifically, the original rewards are accumulated and only an episodic reward is given in the final timestep. The agent will receive a 0 reward in all the other timesteps. This sparse setting makes the tasks much more challenging to solve. Contemporary RL algorithms will usually suffer from the sparse rewards and deliver unsatisfactory performance. We aim to use these sparse variants to study whether our RAPID can effectively capture the sparse rewards.

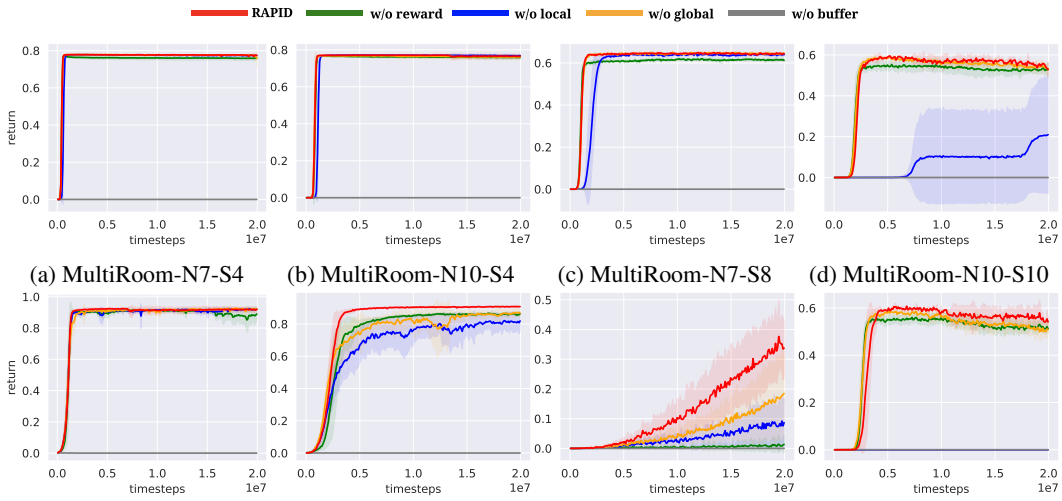


(a) Walker2d (b) Hopper (c) InvertedPendulum (d) Swimmer

Figure 12: Rendering of Mujoco environments.

C ABLATIONS

Figure 13 shows the learning curves of RAPID against different ablations. Without local scores, we observe significant performance drop on challenging environments in terms of sample efficiency and final performance, i.e., on MultiRoom-N7-S8, MultiRoom-N10-S10, MultiRoom-N12-S10, KeyCorridor-S3-R3, and KeyCorridor-S4-R3. This suggests that the proposed local score plays an important role in encouraging exploration. We also observe that without buffer, the agent fails to learn a good policy. Naively using the proposed scores as intrinsic reward will harm the performance. Using the extrinsic reward and the global score to rank the episodes also contribute in the challenging environments, such as KeyCorridor-S4-R3. Therefore, the three proposed scoring methods may be complementary and play different roles in different environments.



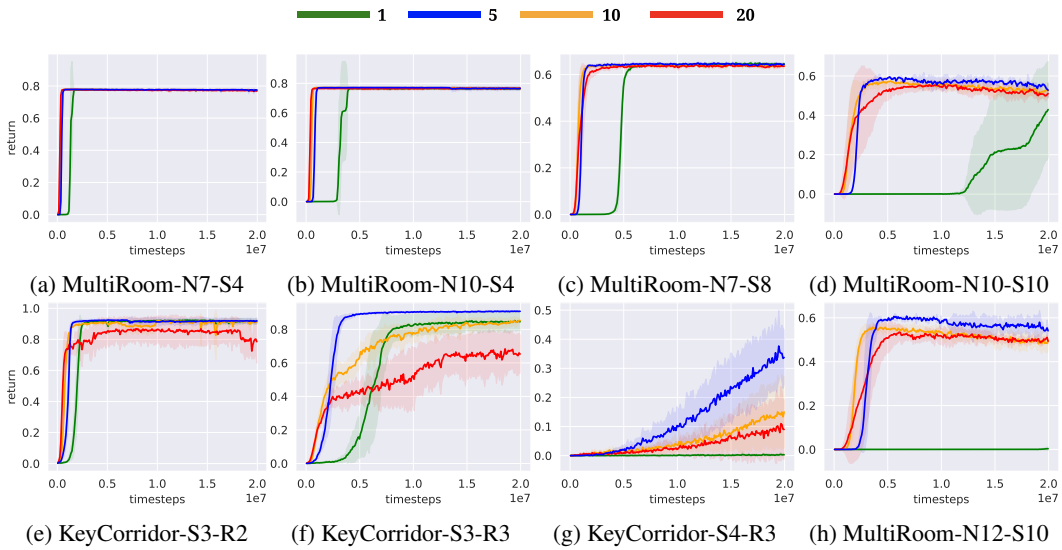
(a) MultiRoom-N7-S4 (b) MultiRoom-N10-S4 (c) MultiRoom-N7-S8 (d) MultiRoom-N10-S10

(e) KeyCorridor-S3-R2 (f) KeyCorridor-S3-R3 (g) KeyCorridor-S4-R3 (h) MultiRoom-N12-S10

Figure 13: Learning curves of RAPID and the ablations

D FULL ANALYSIS RESULTS

D.1 IMPACT OF THE NUMBER OF UPDATE STEPS

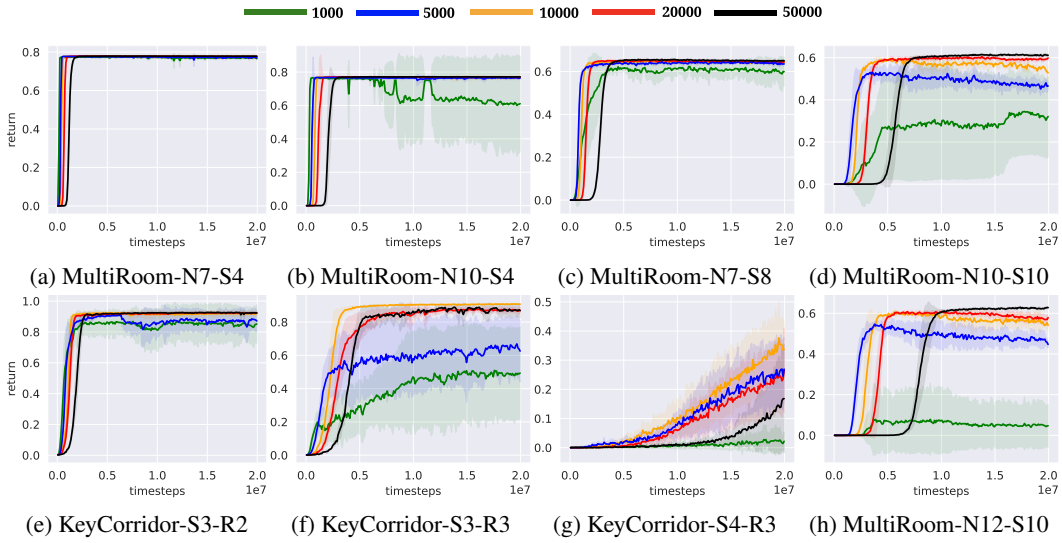


(a) MultiRoom-N7-S4 (b) MultiRoom-N10-S4 (c) MultiRoom-N7-S8 (d) MultiRoom-N10-S10

(e) KeyCorridor-S3-R2 (f) KeyCorridor-S3-R3 (g) KeyCorridor-S4-R3 (h) MultiRoom-N12-S10

Figure 14: Impact of training steps S on MiniGrid environments.

D.2 Impact of Buffer Size

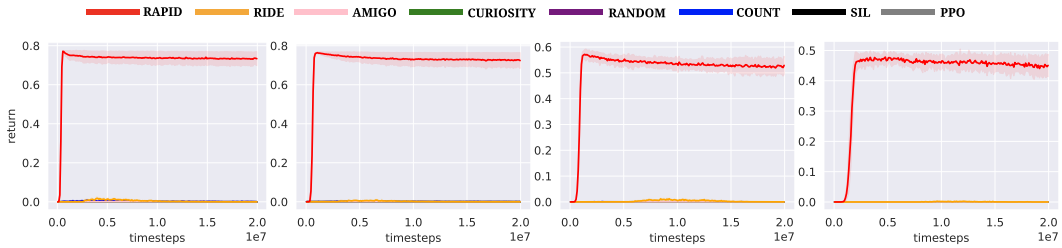


(a) MultiRoom-N7-S4 (b) MultiRoom-N10-S4 (c) MultiRoom-N7-S8 (d) MultiRoom-N10-S10

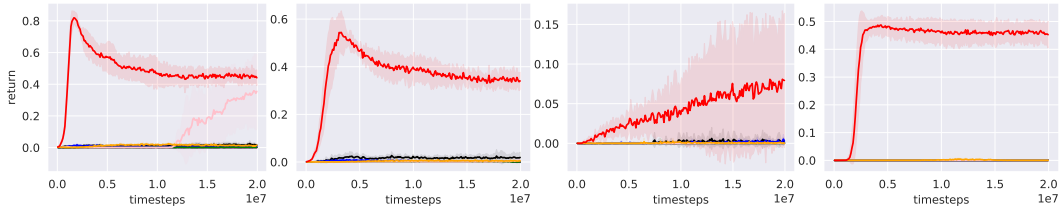
(e) KeyCorridor-S3-R2 (f) KeyCorridor-S3-R3 (g) KeyCorridor-S4-R3 (h) MultiRoom-N12-S10

Figure 15: Impact of buffer size D on MiniGrid environments.

D.3 PURE EXPLORATION ON MINIGRID



(a) MultiRoom-N7-S4 (b) MultiRoom-N10-S4 (c) MultiRoom-N7-S8 (d) MultiRoom-N10-S10

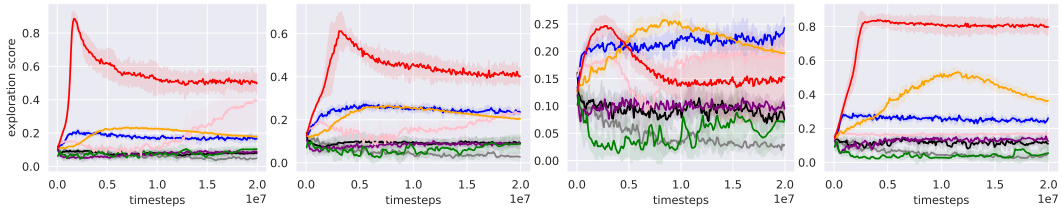


(e) KeyCorridor-S3-R2 (f) KeyCorridor-S3-R3 (g) KeyCorridor-S4-R3 (h) MultiRoom-N12-S10

Figure 16: Extrinsic rewards achieved by RAPID and baselines with pure exploration

D.4 LOCAL EXPLORATION SCORE OF PURE EXPLORATION ON MINIGRID

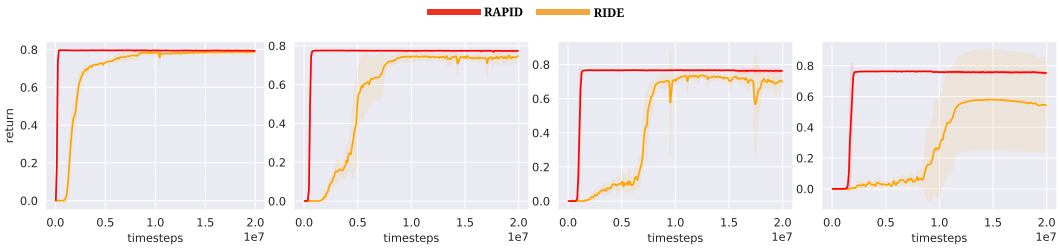
(a) MultiRoom-N7-S4 (b) MultiRoom-N10-S4 (c) MultiRoom-N7-S8 (d) MultiRoom-N10-S10



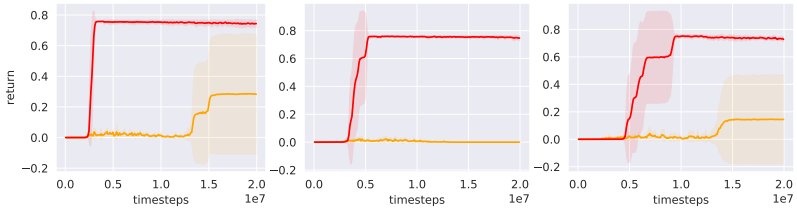
(e) KeyCorridor-S3-R2 (f) KeyCorridor-S3-R3 (g) KeyCorridor-S4-R3 (h) MultiRoom-N12-S10

Figure 17: Local exploration scores achieved by RAPID and baselines with pure exploration

E LEARNING CURVES WITH MORE ROOMS



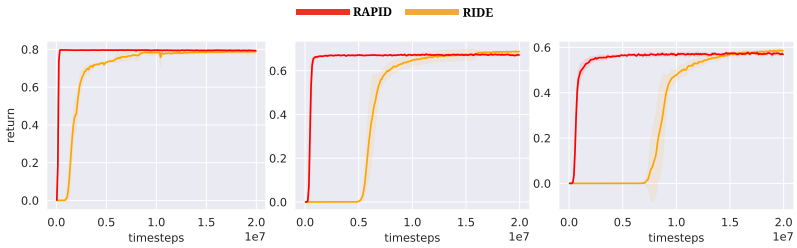
(a) MultiRoom-N4-S4 (b) MultiRoom-N8-S4 (c) MultiRoom-N12-S4 (d) MultiRoom-N16-S4



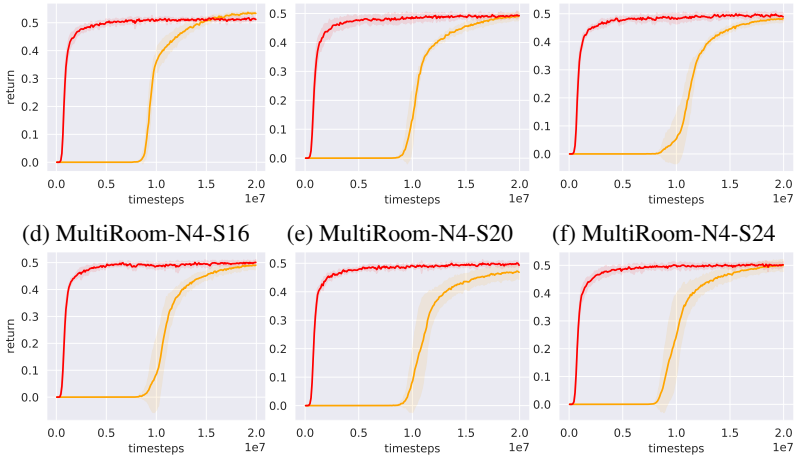
(e) MultiRoom-N20-S4 (f) MultiRoom-N24-S4 (g) MultiRoom-N28-S4

Figure 18: The learning curves with more rooms

F LEARNING CURVES WITH LARGE ROOMS



(a) MultiRoom-N4-S4 (b) MultiRoom-N4-S8 (c) MultiRoom-N4-S12



(d) MultiRoom-N4-S16 (e) MultiRoom-N4-S20 (f) MultiRoom-N4-S24

(g) MultiRoom-N4-S28 (h) MultiRoom-N4-S32 (i) MultiRoom-N4-S36

Figure 19: The learning curves with larger rooms sizes.

G ABLATION AND ANALYSIS OF MINIWORLD-MAZE

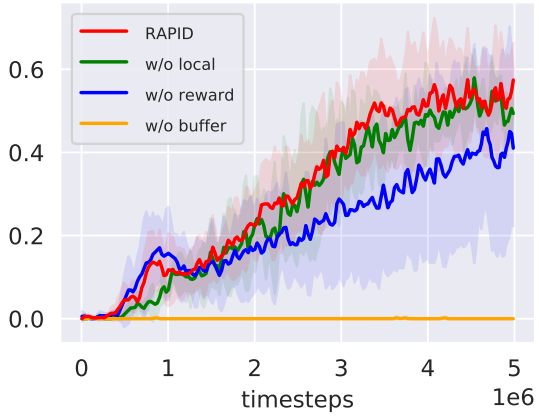


Figure 20: Learning curves of RAPID and the ablations on MiniWorld Maze. We observe minor performance drop when removing the local score, substantial performance drop when removing extrinsic rewards or the buffer.

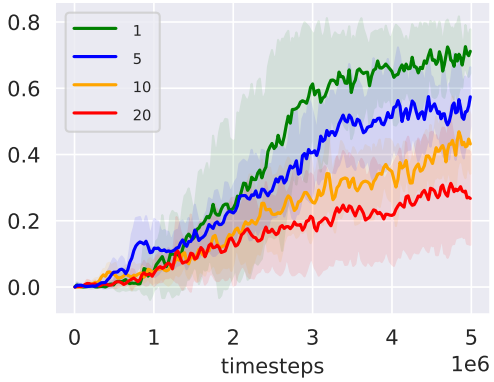


Figure 21: Impact of training steps on MiniWorld Maze.

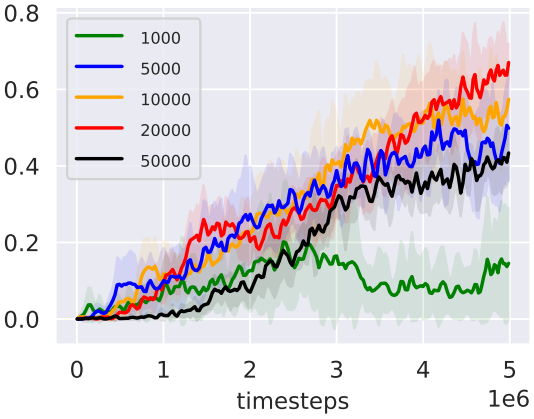


Figure 22: Impact of buffer size on MiniWorld Maze.

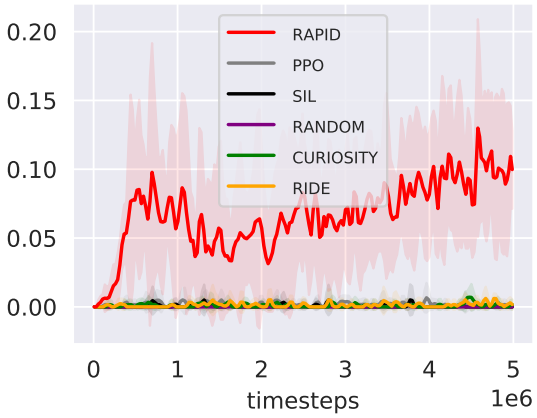


Figure 23: Extrinsic rewards achieved by RAPID and baselines on MiniWorld Maze with pure exploration.

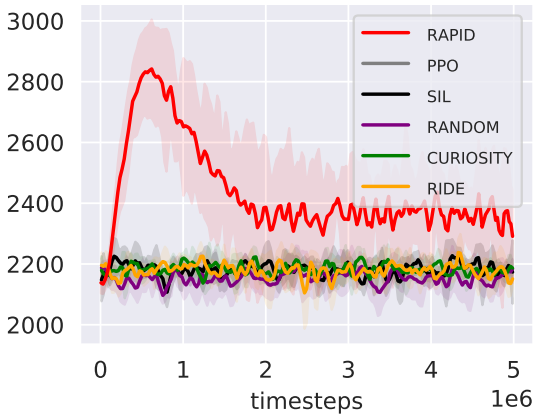


Figure 24: Local exploration scores achieved by RAPID and baselines on MiniWorld Maze with pure exploration.

H Comparison with Storing Entire Episodes in the Buffer

A variant of RAPID is to keep the entire episodes in the buffer instead of state-action pairs. The intuition of keeping the whole episode is that it is possible that a particular state-action pair may only be good in terms of exploration, in the context of the rest of the agent's trajectory in that episode. To test this variant, we force the buffer to keep the entire episode by allowing the episode at the end of the buffer to exceed the buffer size. For example, given that the buffer size is 5000, if the length of the end episode is 160 and the number of state-action pairs in the buffer is 5120, we do not cut off the end episode and exceptionally allow the buffer to store 5120 state-action pairs. In this way, we ensure that the entire episodes are kept in the buffer. We run this experiment on MiniGrid-MultiRoom-N7-S8-v0 (Figure 25). We do not observe a clear difference in the learning curves.

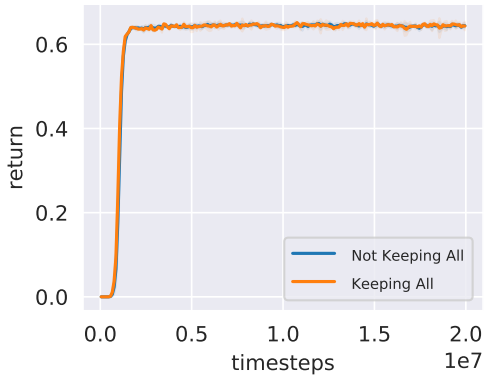
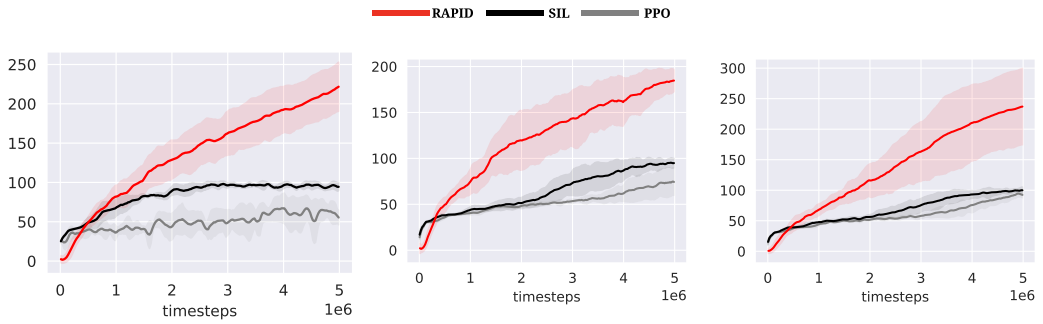


Figure 25: Comparison of keeping all the state-action pairs of an episode and not keeping all the state-action pairs (i.e., a fixed buffer size) on MultiRoom-N7-S8. The experiments are run 5 times with different random seeds. We observe no clear difference in the learning curves of these two implementations.

I Results on Procedurally-Generated Swimmer

The standard MuJoCo environments are singleton. Here, we modify the Swimmer-v2 environment to make it procedurally-generated. Specifically, we make the environment in each episode different by modifying the XML configuration file of MuJoCo engine in every new episode. We consider two environmental properties, i.e., density and viscosity. Density is used to simulate lift and drag forces, which scale quadratically with velocity. Viscosity is used to simulate viscous forces, which scale linearly with velocity. In the standard Swimmer-v2 environment, the density and the viscosity are fixed to 4000 and 0.1, respectively. In the first experiment, we uniformly sample the density in $[2000, 4000]$ in each new episode to make it procedurally-generated, denoted as Density-Swimmer. Similarly, in the second environment, we uniformly sample the velocity in $[0.1, 0.5]$, denoted as Velocity-Swimmer. These two variants make RL training more challenging since the agent needs to learn a policy that can generalize to different densities and velocities. Similarly, we accumulate the rewards to the final timestep of an episode to make the environments sparse. Both environments are provided in our code for reproducibility. The results are reported in Figure 26. We observe that all methods learn slower on these two variants. For example, RAPID only achieves around 190 return in Density-Swimmer, while RAPID achieves more than 200 return in Swimmer. Similarly, SIL reaches around 100 return after around 5×10^6 timesteps in Density-Swimmer, while it achieves around 100 return after around 3×10^6 timesteps in Swimmer. Nevertheless, RAPID can discover very good policies even in these challenging procedurally-generated settings.

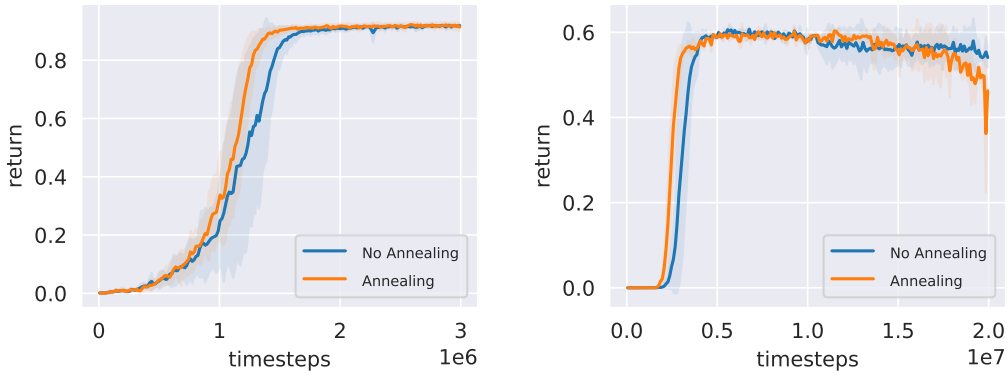


(a) Swimmer (b) Density-Swimmer (c) Velocity-Swimmer

Figure 26: (a) is the singleton swimmer environment. In (b) the density is procedurally-generated. In (c) the velocity is procedurally-generated. All the methods tend to learn slower in the procedurally-generated settings. Nevertheless, RAPID is still able to discover good policies in these challenging variants.

J Discussions of Annealing

In our experiments, we find that annealing the updating step to 0 sometimes helps improve the sample efficiency. However, in some environments, annealing will have a negative effect on the final performance. To show the effect of annealing, we plot learning curves with or without annealing on MiniGrid-KeyCorridorS3R2-v0 and MiniGrid-MultiRoom-N12-S10-v0 in Figure 27. We observe that annealing can help in KeyCorridor-S3-R2. The possible explanation is that, after the agent navigates the goal, it has access to the reward signal. the PPO objective is sufficient to train a good policy with the extrinsic rewards. The imitation loss may harm the performance since it could be better to perform exploitation rather than exploration at this stage. However, in MultiRoom-N12-S10, annealing will harm the performance. In particular, the performance drops significantly in the final stage when the updating frequency of imitation loss approaches zero. The possible reason is that MultiRoom-N12-S10 needs very deep exploration to find the goal. Even though the PPO agent can navigate the goal, it may easily suffer from insufficient exploration when it encounters a new environment. As a result, the agent needs to keep exploring the environment in order to generalize. Introducing imitation loss will ensure that the agent keeps exploring the environments.



(a) KeyCorridor-S3-R2 (b) MultiRoom-N12-S10

Figure 27: Annealing versus not annealing. Annealing is helpful in KeyCorridor-S3-R2. However, the policy fails to converge with annealing in MultiRoom-N12-S10.