Using Discrete Overlapping Partitions for Count-Based Exploration

Anonymous authors
Paper under double-blind review

Keywords: Exploration, Discrete representation, Intrinsic reward

Summary

Exploration remains an open problem in reinforcement learning. Ideally, a useful exploration method should efficiently explore sparse reward environments, scale to large environments, and be simple to implement. Most agents use random-based methods like ϵ-greedy due to their simplicity and low computational cost. However, these methods struggle in sparse reward settings and take a long time to converge. Count-based methods encourage exploration in less-visited areas but do not scale well to large environments. Extensions to count-based methods to work with function approximation improve performance in complex environments like Montezuma's Revenge but are rarely used due to their computational and implementation complexity. We propose a new method that achieves all three desiderata simultaneously. Our exploration method handles large environments by maintaining counts within multiple overlapping partitions of states to derive exploration bonuses. We evaluate our algorithms on three continuous observation environments where count-based methods cannot be applied including MiniGrid DoorKey with image-based observations.

Contribution(s)

1. We introduce a novel, yet simple, method for exploration that uses overlapping partitions of the state space in order to maintain counts, and then derives an exploration bonus from aggregating these counts to encourage exploration to less-visited states.
   Context: Prior work has explored extending count-based exploration methods to large environments, most notably pseudo-counts (Bellemare et al., 2016), though these methods are rarely used due to their complexity. In comparison, our method involves obtaining a discrete representation of the environment, maintaining a count for each cell, and computing the intrinsic reward at each timestep. In this paper, we use tile-coding and VQ-VAE to obtain discrete representations of the environment, but other methods could also be used, such as the representations learned in DreamerV2 (Hafner et al., 2020) or fuzzy tiling activation on the last layer of a neural network (Pan et al., 2019).
2. We show that the new method can explore in sparse reward settings and scale to large environments that use complex function approximators. We use continuous river-swim as a known hard exploration environment and the MiniGrid DoorKey environment as an example image-based domain that uses complex function approximation.
   Context: We build on top of PPO (Schulman et al., 2017) by adding in our exploration bonuses into the optimized return. We compare to the most common approaches to exploration in PPO: random exploration such as epsilon-greedy and entropy regularization on the policy gradient loss. We do not compare to pseudo-count methods as the computational and implementation overhead of these methods make them already very rarely used (Wurman et al., 2022; Bellemare et al., 2020; Berner et al., 2019).

Using Discrete Overlapping Partitions for Count-Based Exploration

Anonymous authors
Paper under double-blind review

Abstract

Exploration remains an open problem in reinforcement learning. Ideally, a useful exploration method should efficiently explore sparse reward environments, scale to large environments, and be simple to implement. Most agents use random-based methods like ϵ-greedy due to their simplicity and low computational cost. However, these methods struggle in sparse reward settings and take a long time to converge. Count-based methods encourage exploration in less-visited areas but do not scale well to large environments. Extensions to count-based methods to work with function approximation improve performance in complex environments like Montezuma's Revenge but are rarely used due to their computational and implementation complexity. We propose a new method that achieves all three desiderata simultaneously. Our exploration method handles large environments by maintaining counts within multiple overlapping partitions of states to derive exploration bonuses. We evaluate our algorithms on three continuous observation environments where count-based methods cannot be applied including MiniGrid DoorKey with image-based observations.

1 Introduction

Exploration is a fundamental challenge in reinforcement learning. What experience is gathered is often as important as the process of learning from it or generalization to new experience. One primitive approach is to use random exploration, e.g., a random action is selected with probability ϵ at every timestep, or an entropy regularization term is included in the training loss to encourage the policy to remain sufficiently random. These approaches often perform extremely poorly in so-called hard exploration settings, most often characterized by sparse rewards. In such settings, the agent relies entirely on this action randomization to stumble on informative rewards before any of the rest of the learning apparatus can make progress.

An early improvement to random exploration has been count-based approaches. These are often motivated under the principle of optimism under uncertainty, where state-action counts can be used to construct exploration bonuses that act to create upper confidence bounds on value estimates (e.g., Strehl & Littman, 2008). These count-based methods, though, are not directly applicable in very large or continuous state and action spaces, where an agent may never revisit a state during typical lengths of trials. In such cases, alternative methods that are friendlier to function approximation have been proposed. For example, Density-based pseudo-counts (Bellemare et al., 2016) construct an exploration bonus from a density model learned alongside a value function. Random network distillation (Burda et al., 2018) creates a bonus based on the error of predicting the output of randomly initialized fixed network of the state input — the prediction improves with observations, decaying the bonus like counts. It would seem we have an ideal answer that is function approximation friendly.

However, celebrated results such as GT-Sophy (Wurman et al., 2022) outperforming professional Gran Turismo drivers or navigation of stratospheric balloons (Bellemare et al., 2020) use simplistic random exploration (entropy regularization and random-walk policies, respectively). Furthermore, Dreamer v3 (Hafner et al., 2023), a general-purpose algorithm achieving state-of-the-art results across many tasks, also eschews directed exploration for simple random exploration from entropy regularization. We propose that another consideration is the simplicity of the exploration method. Pseudo-counts and random-network distillation require significant computational overhead with training a wholly separate network (or two) in addition training of a value or policy network. They also are complicated to implement and correspondingly complicated to make work. Dabney et al. (2020) argues that the complexity of many exploration methods may even limit their generality, suggesting that is the reason ϵ-greedy remains so common.

In summary, there are three desiderata for exploration: effective directed exploration especially in sparse reward settings, function approximation friendliness, and simplicity in implementation, computation, and applicability. In this work we propose a directed exploration approach nearly as simple as count-based methods for tabular function approximation while remaining friendly to common large-scale function approximation schemes, achieving all three desiderate.

Our approach exploits two things. First, many modern function approximation schemes are observing considerable benefit in constructing discrete, but combinatorial, representations. For example, Dreamer (Hafner et al., 2020; 2023) builds discrete states for its world model made up of 32 variables each able to take on 32 discrete values. Each of these easily countable. VQ-VAE (Van Den Oord et al., 2017) gives down-scaled discrete representations of an image, where each latent pixel takes on finite discrete index values; again, easily countable. Fuzzy tiling activations (Pan et al., 2019) provide a discretized final layer representation, aimed at making learning more stable to covariate shift, but, also easily countable. These discrete combinatorial representations, can be thought of as providing multiple overlapping partitions of the state space: each variable partitions the states by the values it can take. These will be what we count. Second, we aggregate the counts associated with each variable to build a single associated count for a state and then use this as a traditional exploration bonus.

We evaluate this approach on simple but challenging toy exploration problems with continuous state spaces and finally on a MiniGrid (Chevalier-Boisvert et al., 2024) sparse reward task using image-based observations. We show that our *simple* count-based method is both *function approximation friendly* and *effective at directing exploration* in sparse reward settings.

2 Background

We begin by introducing the reinforcement learning (RL) problem setting and discuss two popular solution methods, Q-learning (Watkins & Dayan, 1992) and Proximal Policy Optimization (PPO; Schulman et al., 2017), which we use alongside our exploration method. We then present the common exploration strategies associated with these methods, specifically ϵ-greedy for Q-learning and entropy regularization for PPO. We also describe two discretized state representation techniques, tile-coding and VQ-VAE, which are used in Section 4 to obtain overlapping partitions for our exploration method.

Consider a Markov Decision Process (MDP), defined as a tuple $\langle \mathcal{S}, \mathcal{A}, R, P, \gamma \rangle$ where $\mathcal{S}$ represents the state space, $\mathcal{A}$ represents the action space, $R(s, a)$ is the reward function giving the expected reward for taking action a from state s, $P(s'|s, a)$ is the transition function giving the probability of transitioning from state s to state s' given action a, and γ is the discount factor. In this paper we focus on model-free RL where the transition function P is unknown and is not learned by the agent. The environment can be finite and discrete, where $|\mathcal{S}| = d_{\mathcal{S}}$ and $|\mathcal{A}| = d_{\mathcal{A}}$, or it can be continuous, where $\mathcal{S} \subseteq \mathbb{R}^{d_{\mathcal{S}}}$ and $\mathcal{A} \subseteq \mathbb{R}^{d_{\mathcal{A}}}$. When we define counts on state-action pairs, we focus on the case where the state and action space is finite and discrete. When we introduce our new approach we will allow the state space to either be continuous or discrete (and large), but the action space for simplicity will remain discrete.

The goal of an RL agent is to take actions to maximize the discounted sum of rewards, $\sum_{t=1}^{\infty} \gamma^{t-1} r_t$, it receives over time where $\gamma \in [0, 1)$ (Sutton, 2018). We can represent the agent's choice of action with a policy $\pi(s)$ giving a probability distribution over $\mathcal{A}$ for state s. To quantify the the performance of policy π, we define its state-action value or Q-value as:

$$Q^{\pi}(s, a) \equiv \mathbb{E}^{\pi} \left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \middle| s_0 = s, a_0 = a \right], \tag{1}$$

where $\mathbb{E}^{\pi}$ denotes the expected value over all possible trajectories following π. A common goal for RL is to learn an optimal policy $\pi^* = \arg\max_{\pi} Q^{\pi}(s, a)$.

2.1 Q-learning

One common approach to RL is to estimate a value function. The Q-value, can be calculated using the recursive Bellman equation

$$Q^{\pi}(s, a) = R(s, a) + \gamma \sum_{s', a'} P(s'|s, a)\pi(a'|s')Q^{\pi}(s', a'), \tag{2}$$

which is the basis for many learning algorithms, including Q-learning. Q-learning is a temporal difference (TD) control algorithm where Q is updated by computing the TD error, the difference in the Bellman equation given a sample of the next state s' and assuming a' is being selected greedily with respect to the current Q estimate. Using α as a step size, the Q estimate is updated at time t by,

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]. \tag{3}$$

Here, we assume we have a finite action and state space, and the Q estimates are stored as a table. In order to guarantee convergence to the optimal policy, all states and actions need to be visited infinitely often, i.e., we need an effective exploration method. This is often achieved through selecting actions ϵ-greedy with respect to the current Q estimates.

2.2 Proximal Policy Optimization (PPO)

Another common approach is to represent and learn a policy directly. Let $\pi_\theta(a|s)$ be a parameterized representation of the policy with parameters θ, e.g., using a neural network. The goal becomes to directly update θ to take actions that accumulate more reward, typically through gradient updates. One such approach is PPO (Schulman et al., 2017), which also learns a value function used to reduce variance in the policy updates. PPO collects experience through interactions with the environment and optimizes an objective function using minibatch updates from multiple trajectories. PPO's objective to maximize can be written as:

$$J_t(\theta) = \mathbb{E}_t \left[J_t^{\text{CLIP}}(\theta) - c_1 L_t^{\text{VF}}(\theta) + c_2 H[\pi_\theta] \right], \tag{4}$$

where c_1, c_2 are coefficients, J_t^{CLIP} is the policy improvement objective (clipped to prevent the policy from changing too drastically on each update, i.e., keep it "proximal"), L_t^{VF} is the value function error and $H[\pi_\theta]$ is an entropy regularizer. Our main focus is the entropy regularizer, which serves as PPO's main exploration mechanism.

2.3 Exploration Bonuses

An influential approach to exploration is the principle of Optimism in the Face of Uncertainty (OFU). Instead of greedily choosing actions based solely on estimated rewards (with or without undirected randomness), the agent assumes that less visited states and actions may have a higher value than estimated. This is typically done with some form of exploration bonus added to the environment reward to drive a greedy policy to seek less visited states and actions. As the agent explores and visits

more states and actions, it reduces its uncertainty and the associated exploration bonus. The Upper Confidence Bound (UCB; Auer, 2002) algorithm belongs to the OFU family. It was introduced in the multi-armed bandit setting, but has also been used extensively in the MDP setting (e.g., **?**Bellemare et al., 2016). The agent operates as if the observed reward is $r_t = R(s_t, a_t) + \beta R^+(s_t, a_t)$ where $R^+(s_t, a_t)$ denotes the exploration bonus, and β is the coefficient balancing exploration and exploitation. UCB suggests the following count-based exploration bonus:

$$R^+(s, a) = \sqrt{\frac{2 \log N(s)}{N(s, a)}} \tag{5}$$

where $N(s, a)$ is the state-action pair visitation count and $N(s) = \sum_a N(s, a)$. Maintaining counts for every state-action pair is not applicable in large or continuous environments. Not only is it intractable to store such counts, but an agent may never revisit a state during typical lengths of trials, and so will not effectively direct exploration. Density-based pseudo-counts (Bellemare et al., 2016) and random network distillation (Burda et al., 2018) are attempts to devise count-like alternatives for Equation 5 that are function approximation friendly. However, these are complex in both implementation and computation, and so do not meet the stated desiderata.

2.4 State Discretization

Our approach makes use of a growing trend in RL: learned discretized state representations. In order to have high expressivity, these representations are typically combinatoric, i.e., the state is represented as the composition of a set of discrete variables. Each discrete variable of a state, $v_i(s) \in [n] \equiv \{1, \ldots, n_i\}$, can be thought of as partitioning states into a finite number of cells indexed by positive integers. And the overall representation $\phi(s) = v_1(s) \times \ldots \times v_n(s)$ is the Cartesian product of these multiple overlapping discrete partitions. Such a combinatoric representation is not new: CMACs (Albus, 1975) and tile-coding (Sutton, 1995) are examples of such representations. In tile-coding, a grid or tiling is defined on top of a vector representation of the state so as to partition the states into tiles. By offsetting the tiling by different amounts, multiple overlapping partitions can be created, where the state is represented by the set of discrete tiles that contains it in each tiling. Large tiles then give a sense of generalization, as states that are distant will share some tiles in the representation, while a large number of tilings gives expressivity as nearby states can still be in different tiles for some tiling.

The recent trend is to learn such discrete representations. Dreamer (Hafner et al., 2020; 2023) is one such example where a state's learned latent representation is a binary matrix with 32 rows of one-hot encoded vectors of length 32. Each row, like tile-coding, partitions the state-space into 32 cells, with the one-hot encoding identifying the cell. Expressivity comes from combinatorics of representing the state with 32 such discrete partitions. Vector Quantized Variational Auto Encoders (VQ-VAE; Van Den Oord et al., 2017) is another example, and one we will use in our experiments. A VQ-VAE consists of an encoder, a quantization step, and a decoder, aimed at finding a latent representation of images that ideally respects an image's compositionality. By passing the image input to the encoder we can learn a representation of the input $z_e(x) \in \mathbb{R}^{n \times m}$, which gives n vectors of length m. These vectors are compared to a dictionary of embedding vectors $e \in \mathbb{R}^{k \times m}$. For vector i, let $v_i \in [k]$ be the row index of the chosen vector of e. These chosen vectors, $z_q \in \mathbb{R}^{n \times m}$, are passed through a decoder to reconstruct the given input. The encoder, decoder, and dictionary of embedding vectors are all learned in VQ-VAE. Notice that the Cartesian product $v_1(s) \times \ldots \times v_n(s)$ gives multiple overlapping partitions of its input, just as with tile-coding. Locally sensitive hashing (Indyk & Motwani, 1998; Bellemare et al., 2013) and fuzzy tiling activations (Pan et al., 2019) are two other examples of fixed and learned discrete representations, respectively.

3 Method

Consider the Continuous Riverswim (Pan et al., 2018) environment (see Figure 15), which expands the discrete 6-state classic environment, to the $[0, 1]$ interval. The environment has two actions:

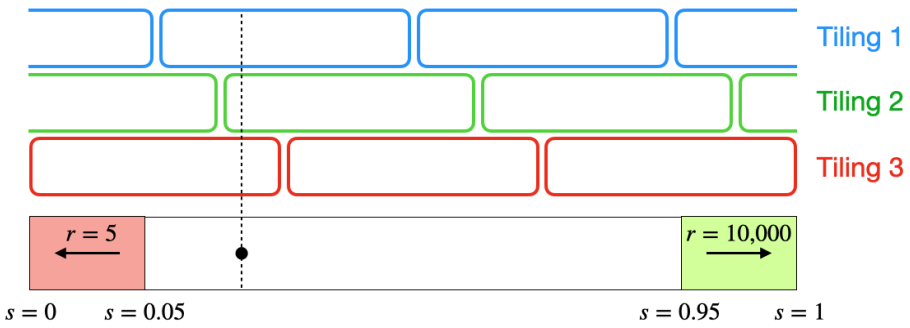


Figure 1: Continuous Riverswim illustration. The red and green regions indicate low- and high-reward areas, respectively. The black dot represents the agent within three overlapping tilings.

LEFT and RIGHT. LEFT moves the agent 0.1 units to the left , while RIGHT moves the agent 0.1 units right (with 30% probability), 0.1 units left (with 10% probability), or keeps it in place (with 70% probability). After moving, the agent's position is perturbed by a sample from a zero-mean Gaussian with standard deviation 0.01. If the agent takes RIGHT from a state ≥ 0.95 it receives a reward of $10,000$; LEFT from a state ≤ 0.05 a reward of 5, otherwise 0; $\gamma = 0.99$. The optimal policy is to always choose RIGHT, however it is difficult and unlikely to observe the large reward doing so, while the small reward for LEFT is easily found. While count-based methods efficiently explore in the discrete version of Riverswim, it cannot be applied in Continuous Riverswim.

Now consider applying tile-coding to the Continuous Riverswim state space, where each partition is just a one-dimensional tiling (see the tiling illustration in Figure 1), and each subsequent tiling is offset to create different boundaries between cells in each partition. A single state can then be represented by the cells from each partition that contains that state. It is these cells, i.e., tiles, that our approach will count, so each state visited leads to incrementing the count of all n cells containing the state. To construct an exploration bonus from the n counts, we use an aggregation function to turn a set of counts into a single count and then apply Equation 5 to the aggregate count. Possibilities for the aggregation function include average, max, and min. The choice of min for the aggregation is suggestive of the count-min-sketch (Cormode & Muthukrishnan, 2005) for estimating element counts with sublinear memory, which also counts cells of multiple overlapping partitions but uses perfect pairwise-independent hash functions as the partitions. The choice of average is suggestive of tile-coding itself, where a value function is represented by the average value of each cell.

Formally, imagine a set of overlapping partitions $\{P_i\}_{i=1\ldots n}$ where each partition $P_i : \mathcal{S} \to [k]$ maps to k possible cells. We can represent the set of all cells with $C = \{(i, j) \mid i \in [n],\ j \in [k]\}$. We keep counts on every cell action pair, $N(c \in C, a \in \mathcal{A})$. Given an aggregation function $A : \mathbb{N}^n \to \mathbb{R}$ we construct a state action count as $N(s, a) \equiv A(\{N(P_i(s), a)\}_{i=1\ldots n})$, which then gets used to create a UCB exploration bonus with Equation 5.

Tile-coding becomes impractical in high-dimensional spaces because the number of required tilings increases exponentially with the number of dimensions to maintain a fixed resolution. Furthermore, as a fixed representation it does not adapt its expressivity to relevant differences in states. Hence, it would not be practical, for example, for image-based state representations. However, the general process can be applied to any discrete representation with multiple overlapping partitions, including the recent trends of learned discrete representations, such as those discussed in Section 2.4. In particular, we will experiment with exploring using counts on the discrete latent spaces produced by VQ-VAE.

4 Results

We evaluate our method across three distinct environments with different characteristics to analyze its effectiveness in various reinforcement learning settings. First, we use the Continuous RiverSwim (Pan et al., 2018) environment, a small continuous environment with a distractor reward that makes it a challenging exploration problem for Q-learning. Second, we evaluate on the Mountain Car (Moore, 1990) environment, a classic control task with sparse rewards, using PPO. Lastly, we evaluate our method on the MiniGrid DoorKey (Chevalier-Boisvert et al., 2024) environment, where the agent receives image-based observations using VQ-VAE representations. We conducted experiments on both the standard DoorKey setting and a modified version with shorter episodes to make the exploration problem harder. Unless otherwise noted, all results are averages over 50 runs and shaded regions show 95% confidence intervals. The exact details of all of the environments as well as the hyperparameters for the experiments are all reported in the supplementary material.

4.1 Continuous Riverswim

Continuous Riverswim is a challenging exploration task that requires a directed exploration strategy. The difficulty of a random walk to reach the large reward and the presence of the small distractor reward increases the difficulty of exploration, as the agent prematurely may seek the suboptimal reward. We trained agents using Q-learning with different exploration methods: ϵ-greedy, state aggregation (discrete representation with no overlapping partitions), and multiple overlapping partitions using tile-coding with two different aggregation functions: average and min. The same discrete representation was used for both learning and exploration bonuses, with the number of tilings and tiles not optimized for either exploration or learning. Figure 2 shows the total undiscounted return over 100,000 steps of learning. We see that using overlapping discrete representations with either of the aggregation functions yields a higher return than state aggregation and ϵ-greedy, as it more reliably found the action leading to the large reward. Amongst the two aggregation functions we see that min is a slower to begin exploiting than average but more reliably exploits the large reward. Note that despite tuning ϵ in ϵ-greedy, testing values $\epsilon \in \{0.1, 0.2, 0.5\}$, none of the runs discovered the large reward, consistently exploiting the suboptimal reward.

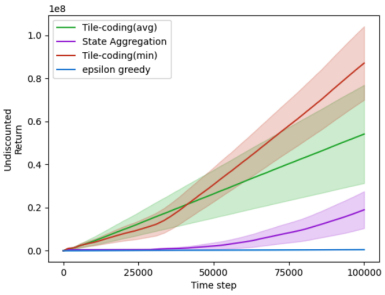


Figure 2: Continuous RiverSwim showing total undiscounted return over 100,000 timesteps for Q-learning with different exploration methods. "Tile-coding(X)" denotes our proposed approach with different aggregation functions. State Aggregation denotes using counts with a single partition of the state space.

4.2 Mountain Car

Mountain Car is a classic control environment with continuous state and sparse reward (rewards are -1 except at episode termination). In this environment, we use PPO for learning as it is known to struggle in Mountain Car, while using tile-coding to obtain discrete state representations for exploration bonuses. Similar to Continuous Riverswim (Section 4.1) we do not tune the number of tilings or tiles. For PPO's policy and value networks, we use an MLP with position and velocity as input and two hidden layers, each containing 64 units, with $tanh$ as the activation function. Figure 3 presents the results for entropy regularization and our count-based method using

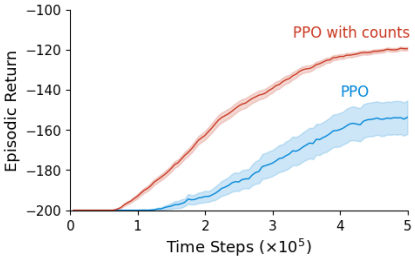


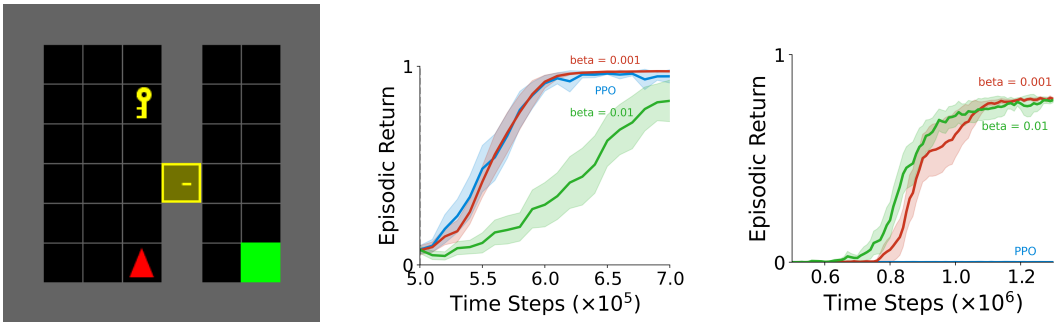
Figure 3: Mountain Car showing episodic return over 500k timesteps for PPO using tile-coding counts and vanilla PPO with entropy regularization.

average as the aggregation function. As shown in Figure 3, PPO with our count-based exploration learns approximately twice as fast as PPO with entropy regularization. Count-based exploration also demonstrates lower variance in the episodic return, as indicated by the shaded confidence intervals. This is likely due to PPO with entropy regularization failing to reliably learn a policy to reach the goal within the training duration on some trials, which demonstrates the weakness of relying on strictly random exploration rather than directed exploration to reduce uncertainty as with exploration bonuses.

4.3 MiniGrid Doorkey

Finally, we test our approach in the MiniGrid DoorKey environment (Chevalier-Boisvert et al., 2024). MiniGrid DoorKey at its hear is a gridworld consisting of two separate rooms divided by a wall and a locked door. The agent's goal is to pick up the key from one room, unlock the door, and reach the green tile in the second room. However, the agent receives an image of the environment as its state observation. The available actions include turning right, turning left, moving forward, picking up an object, and using an object (e.g. using the key to unlock the door). The agent receives a reward of 0 on each timestep, and a reward of $1-0.9\frac{t}{T}$ upon reaching the goal, where t is the current timestep and T is the maximum number of timesteps in an episode. The environment is stochastic: each time the agent selects an action, there is a 10% probability that a different random action will be executed instead. Figure 4a depicts the 8×8 MiniGrid DoorKey environment used in our experiments, illustrating the agent's starting position, goal location, and the key's location. Finally, an episode terminates after 1,000 timesteps if the goal has not been reached. We observed that this makes for an easy exploration policy as a 1,000 step random-walk has a significant chance of reach the goal, which is under 20 steps from anywhere in the environment. We also experimented with a more challenging variant, where episodes were terminated after 100 timesteps, making directed exploration more important.

To encode the environment into multiple overlapping partitions, we train a VQ-VAE model. These representations are later used to train a PPO agent and maintain state visitation counts. We follow the model-free experimental settings of Meyer et al. (2023). A VQ-VAE model is pre-trained with 500k steps of data gathered from a random walk in the environment. The discrete representation was then used as both the input state for PPO's policy and value function, as well as the basis for defining exploration bonuses from the cell counts.



(a) DoorKey Environment (b) 1000 steps per episode (c) 100 steps per episode

Figure 4: Minigrid DoorKey showing the environment (a); episodic return with 1,000 step episodes (b) and 100 step episodes (c). Results averaged over 30 runs. Training shown starting after 500k steps of pre-training the VQ-VAE representation.

Figure 4b presents our results on the standard MiniGrid DoorKey environment, where the maximum number of timesteps per episode is 1,000. Agents were trained for 200k steps, which we show starting from 500k to account for the VQ-VAE pre-training. We compare a PPO agent using entropy regularization for exploration with two PPO agents employing count-based exploration, each with different exploration coefficients β. Our approach with $\beta = 0.001$ and entropy regularization exhibit

similar performance and converge to an optimal policy within 100K from the start of PPO training. However, $\beta = 0.01$ is still learning after 200k steps, which we believe is due to excessive exploration caused by the large exploration bonus coefficient. We believe the agent must spend significantly more time visiting states throughout the environment to sufficiently collapse its uncertainty before the environment reward dominates. State distribution plots in Supplementary Materials S1 confirm this behavior: even after solving the task, the agent with $\beta = 0.01$ continues to explore other regions, reducing overall efficiency. These results show the importance of balancing between exploring enough to identify the optimal policy and not over-exploring, which can delay exploiting with one's learned policy.

What about the harder challenge of shorter episodes, increasing the need to efficiently explore before restarting in the initial state. Figure 4c shows the average episodic return for the same agents. PPO with entropy regularization regularly failed to reach the goal even once in 200k steps, exhibiting no learning at all, as it relied only on a random walk to observe non-zero return. Our approach showed that it could reliably learn good policies in this hard exploration task, with the larger exploration bonus ($\beta = 0.01$) proving to learn slightly faster, again emphasizing the need to balance exploration with the difficulty of the task. This improved exploration was achieved by simple counts on discrete quantities, requiring almost no additional computation and or complexity of implementation.

5 Conclusion

We introduced a simple count-based method for exploration that exploits discrete representations consisting of multiple overlapping partitions of the state space. We show that the method can do efficient exploration in hard exploration problems, such as Continuous Riverswim, as well as being function approximation friendly while using VQ-VAE representations of image-based observations. This method addresses the three desiderata of simple, effective and function approximation friendly. There are a number of future directions that would be valuable to explore. First, a thorough investigation of aggregation functions is needed. Can some theory of count-min-sketch be extended to this setting where representations are not pairwise-independent hashes? Are there situations where min or average perform better? Might soft-min give additional control over the manner of exploration? Furthermore, how does this approach fair when the discrete representation is being learned simultaneously? Ultimately, we see this approach as sufficiently simple and effective to replace the surprising resilience of ϵ-greedy.

References

James S Albus. A new approach to manipulator control: The cerebellar model articulation controller (cmac). *Journal of Dynamic Systems, Measurement, and Control*, 97(3):220–227, 1975.

P Auer. Finite-time analysis of the multiarmed bandit problem, 2002.

Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation. *Advances in neural information processing systems*, 29, 2016.

Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of artificial intelligence research*, 47: 253–279, 2013.

Marc G Bellemare, Salvatore Candido, Pablo Samuel Castro, Jun Gong, Marlos C Machado, Subhodeep Moitra, Sameera S Ponda, and Ziyu Wang. Autonomous navigation of stratospheric balloons using reinforcement learning. *Nature*, 588(7836):77–82, 2020.

Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*, 2018.

Maxime Chevalier-Boisvert, Bolun Dai, Mark Towers, Rodrigo Perez-Vicente, Lucas Willems, Salem Lahlou, Suman Pal, Pablo Samuel Castro, and Jordan Terry. Minigrid & miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks. *Advances in Neural Information Processing Systems*, 36, 2024.

Graham Cormode and Shan Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

Will Dabney, Georg Ostrovski, and André Barreto. Temporally-extended ϵ-greedy exploration. *arXiv preprint arXiv:2006.01782*, 2020.

Danijar Hafner, Timothy Lillicrap, Mohammad Norouzi, and Jimmy Ba. Mastering atari with discrete world models. *arXiv preprint arXiv:2010.02193*, 2020.

Danijar Hafner, Jurgis Pasukonis, Jimmy Ba, and Timothy Lillicrap. Mastering diverse domains through world models, 2023. *URL https://arxiv. org/abs/2301.04104*, 2023.

Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pp. 604–613, 1998.

Edan Meyer, Adam White, and Marlos C Machado. Harnessing discrete representations for continual reinforcement learning. *arXiv preprint arXiv:2312.01203*, 2023.

Andrew William Moore. Efficient memory-based learning for robot control. Technical report, University of Cambridge, Computer Laboratory, 1990.

Yangchen Pan, Muhammad Zaheer, Adam White, Andrew Patterson, and Martha White. Organizing experience: a deeper look at replay mechanisms for sample-based planning in continuous state domains. *arXiv preprint arXiv:1806.04624*, 2018.

Yangchen Pan, Kirby Banman, and Martha White. Fuzzy tiling activations: A simple approach to learning sparse representations online. *arXiv preprint arXiv:1911.08068*, 2019.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

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

Richard S Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in neural information processing systems*, 8, 1995.

Richard S Sutton. Reinforcement learning: An introduction. *A Bradford Book*, 2018.

Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8:279–292, 1992.

Peter R Wurman, Samuel Barrett, Kenta Kawamoto, James MacGlashan, Kaushik Subramanian, Thomas J Walsh, Roberto Capobianco, Alisa Devlic, Franziska Eckert, Florian Fuchs, et al. Outracing champion gran turismo drivers with deep reinforcement learning. *Nature*, 602(7896):223–228, 2022.

Supplementary Materials

The following content was not necessarily subject to peer review.

S1 State distribution

In this section, we present the state distribution of three PPO agents using count-based exploration with $\beta \in \{0.01, 0.001\}$ and PPO with entropy regularization on MiniGrid DoorKey in two variations: the standard version (1,000 maximum steps per episode) and a modified version with a different step limit (1,000 maximum steps per episode). For both environments, we plot the state distribution to visualize how agents explore the state space over time, analyzing visitation frequency and coverage. Additionally, we remove the environmental reward to isolate the effect of intrinsic exploration, allowing us to examine how agents navigate without external incentives.

S2 Hyperparameters

For Continuous Riverswim we swept over $\beta \in \{1, 10, 100, 1000, 10000\}$, $\epsilon \in \{01., 0.2, 0.3, 0.4, 0.5\}$, and $\alpha \in \{0.001, 0.1, 1, 10\}$ for all three methods (ϵ-greedy, state aggregation and tile-coding). Before running the main experiment we tuned our hyperparameters and selected parameters that yielded highest average reward.

In the Mountain Car we sweep over $\alpha \in \{2.5 \times 10^{-3}, 2.5 \times 10^{-4}, 2.5 \times 10^{-5}\}$, the hyperparameters were chosen as standard parameters in classic control problem. The hyperparameters for PPO are shown in Table 1.

Table 1: Hyperparameters for Mountain Car

Hyperparameter	**Value**
PPO mini batch size	128
PPO iterations	4
PPO clipping factor (ϵ)	0.2
PPO value coefficient (c_1)	0.5
PPO entropy coefficient (c_2)	0.001

We use the same hyperparameters as those in Meyer et al. (2023) for PPO and VQ-VAE. We perform a grid search over the step size, $\alpha \in \{3 \times 10^{-3}, 3 \times 10^{-4}, 3 \times 10^{-5}\}$, and β coefficient for the count-based exploration algorithms, $\beta \in \{0.01, 0.001, 0.0001\}$, keeping all other parameters unchanged. For count-based exploration, the *min* function was chosen as the aggregate function, and the best step size for each coefficient was selected. For every coefficient in both variations of the environment $\alpha = 3 \times 10^{-5}$ had the highest performance except $\beta = 0.001$ in the standard version where $\alpha = 3 \times 10^{-4}$ was chosen.

The fixed hyperparameters for PPO and VQ-VAE are listed in Table 2.

S3 Environments

S3.1 Continuous Riverswim

Continuous Riverswim, introduced by (Pan et al., 2018), is a modified version of Riverswim (Strehl & Littman, 2008). On the leftmost side of the environment, a distractor reward encourages the agent to remain there unless sufficient exploration is done, posing a significant challenge. The state space is $s \in [0, 1]$, and the actions are $a \in \{left, right\}$. Once the direction of movement is

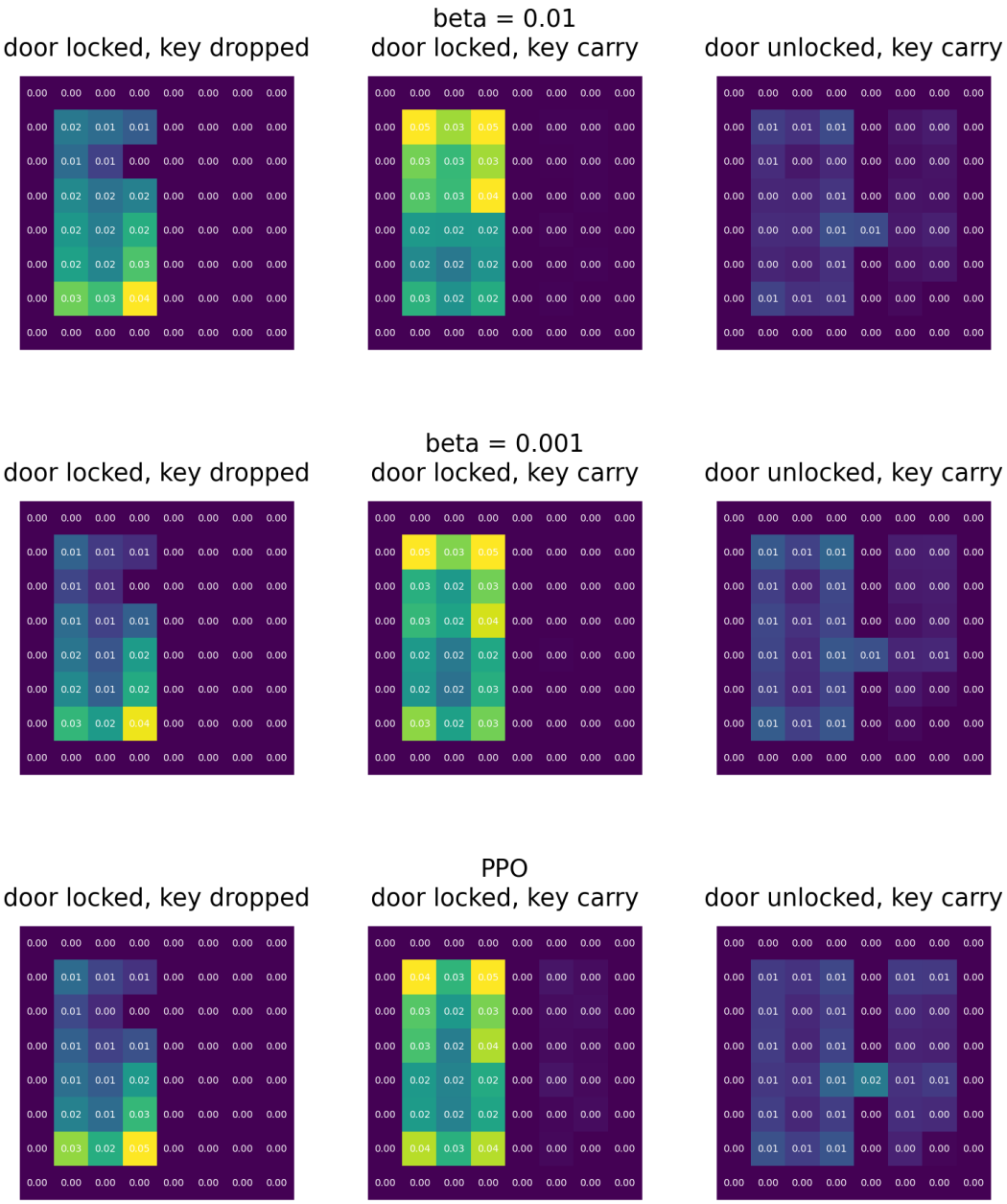


Figure 5: State distribution for standard DoorKey (1000 steps) after 20k steps of training PPO

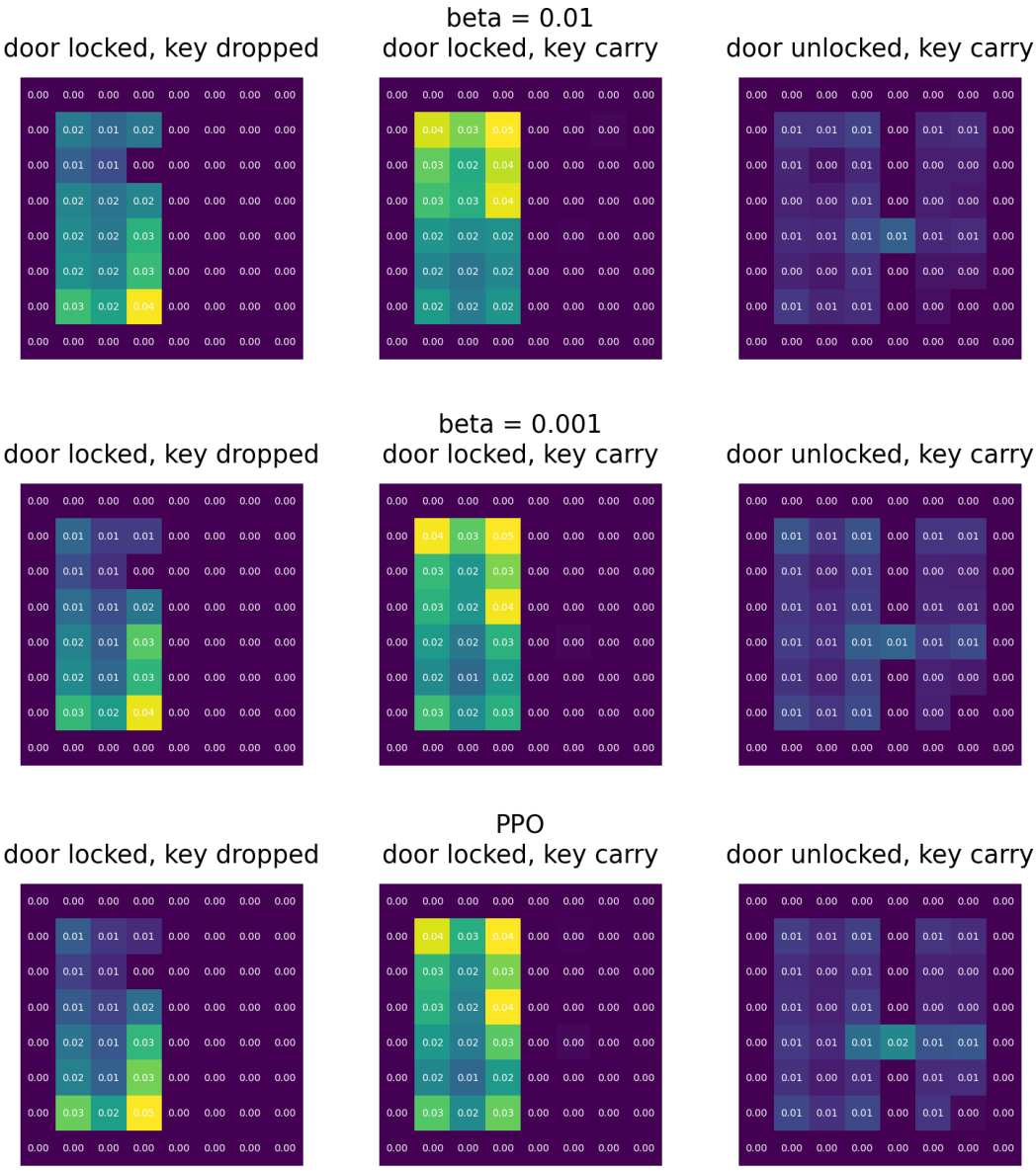


Figure 6: State distribution for standard DoorKey (1000 steps) after 50k steps of training PPO

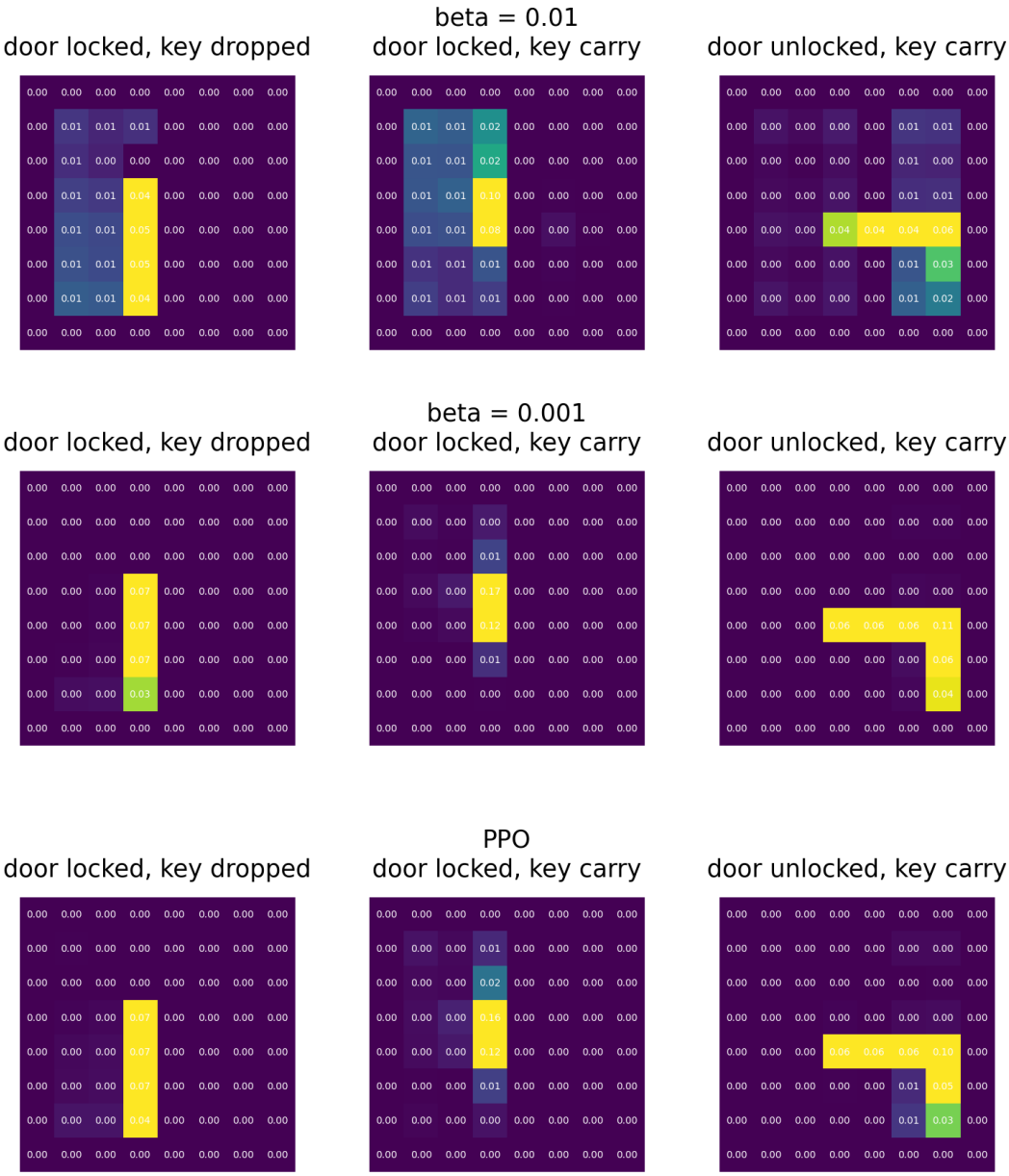


Figure 7: State distribution for standard DoorKey (1000 steps) after 200k steps of training PPO

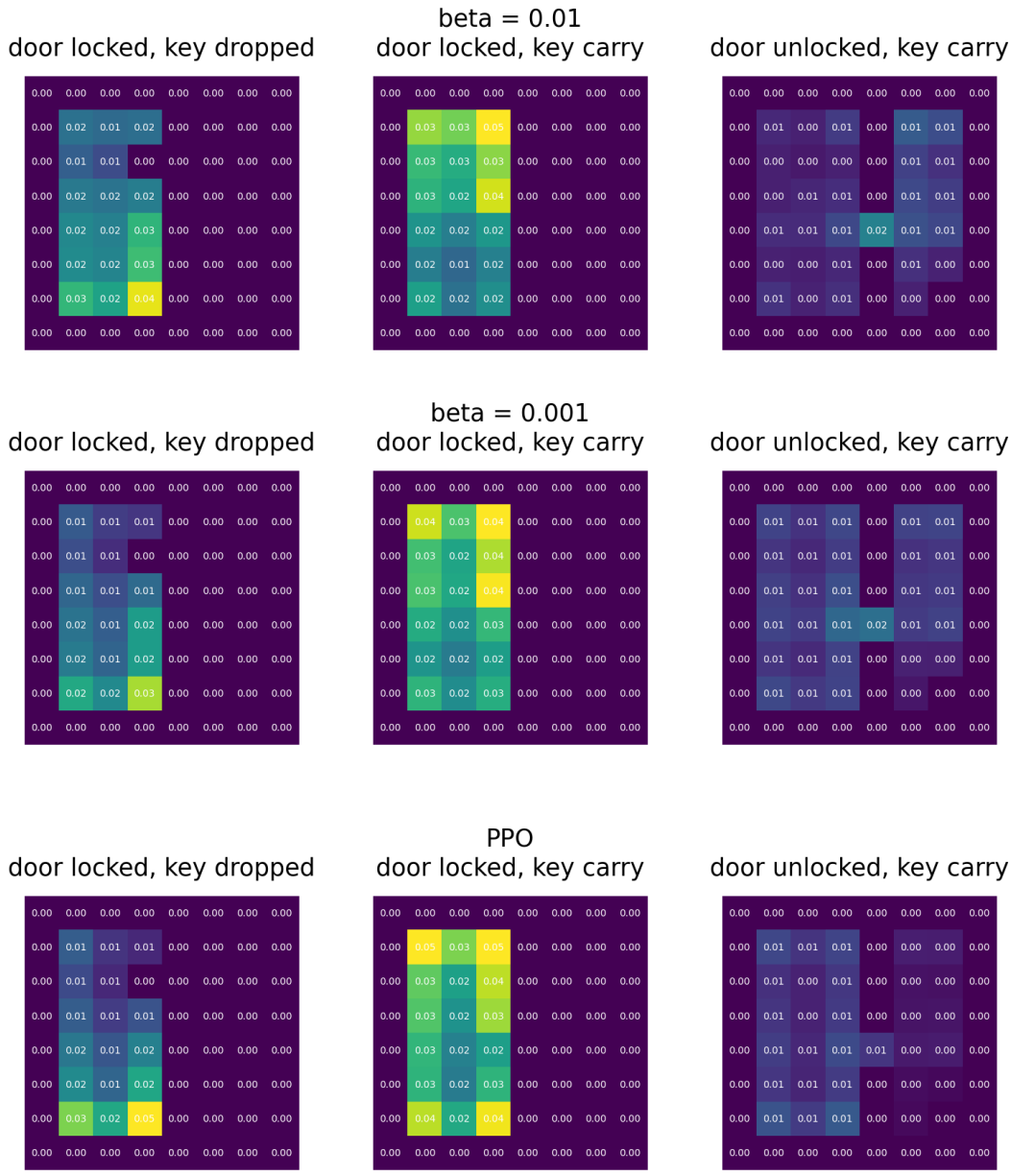


Figure 8: State distribution for standard DoorKey (1000 steps) without reward after 100k steps of training PPO

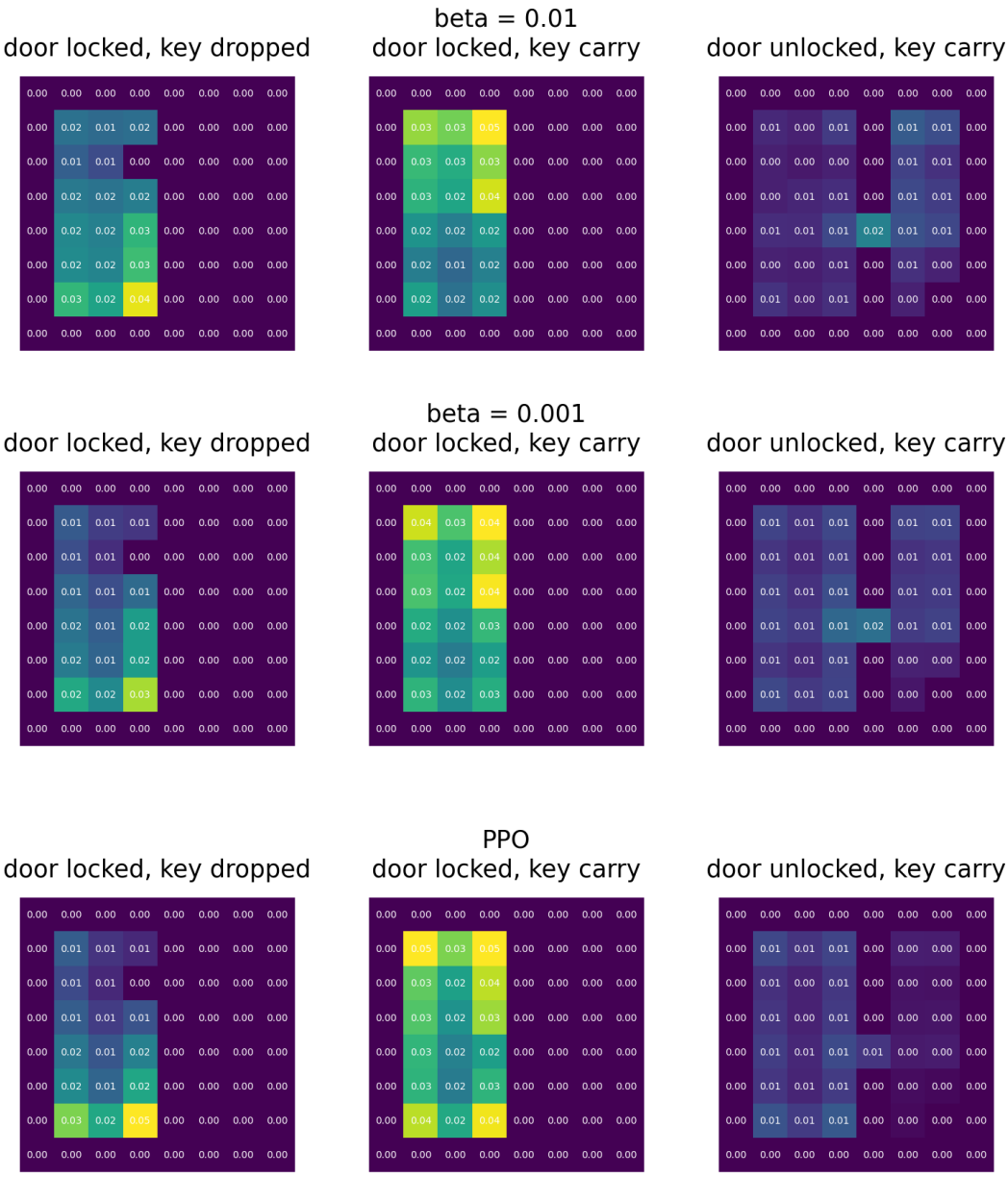


Figure 9: State distribution for standard DoorKey (1000 steps) without reward after 100k steps of training PPO

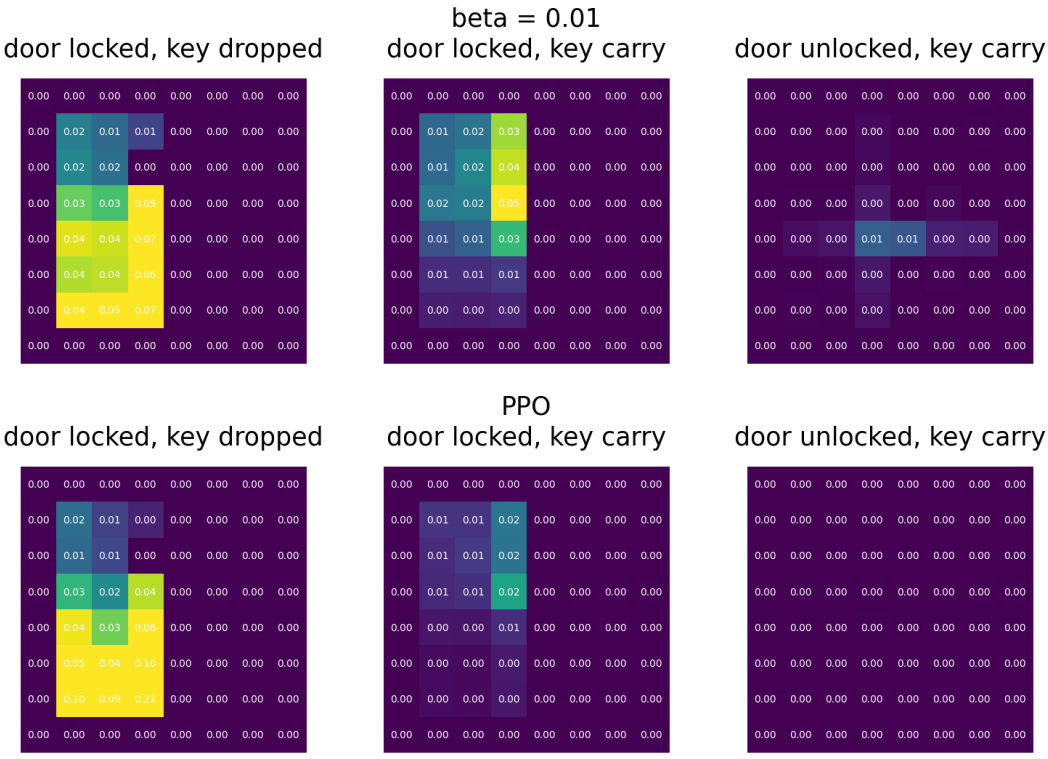


Figure 10: State distribution for modified DoorKey (100 steps) after 200k steps of training PPO

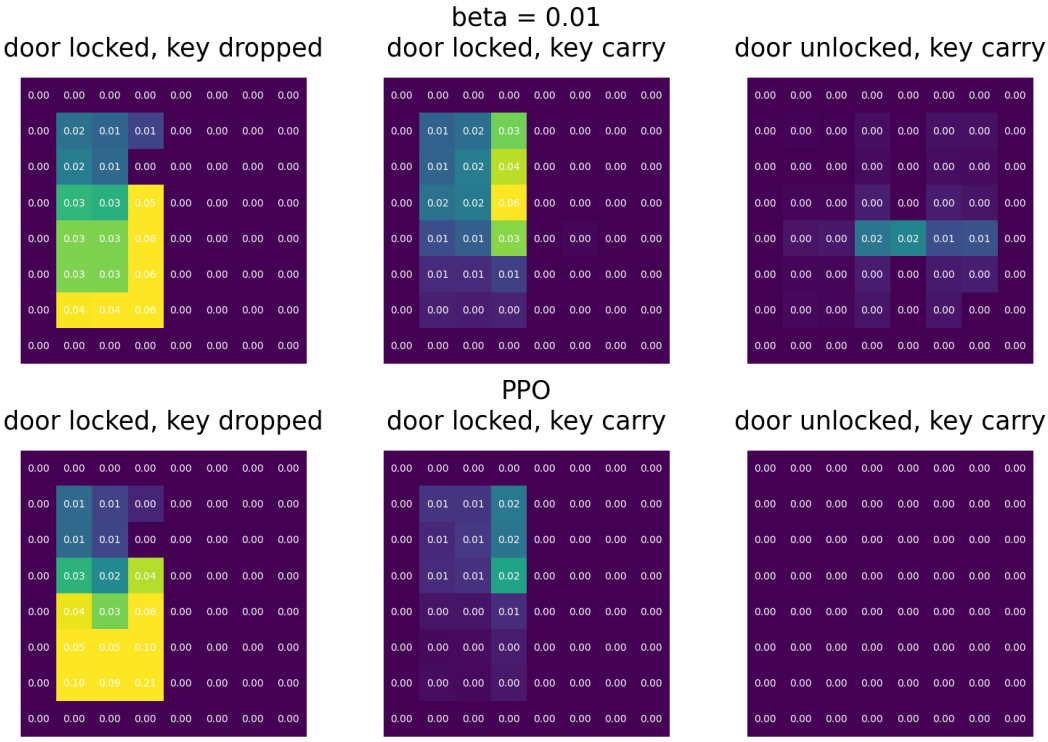


Figure 11: State distribution for modified DoorKey (100 steps) after 300k steps of training PPO

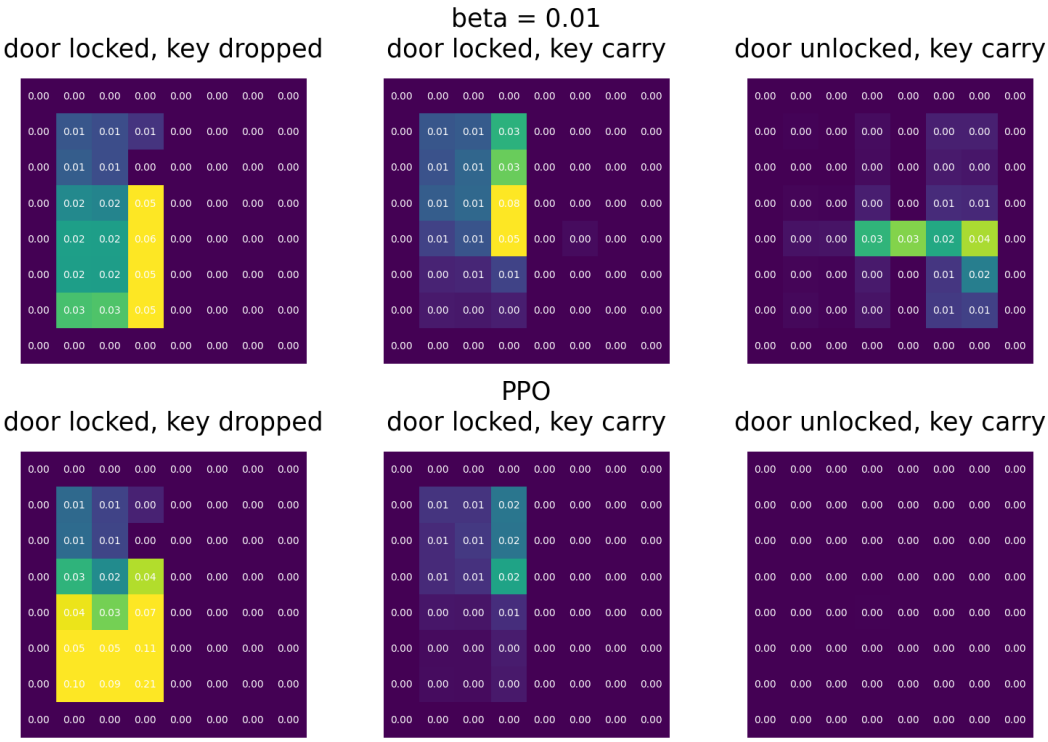


Figure 12: State distribution for modified DoorKey (100 steps) after 500k steps of training PPO

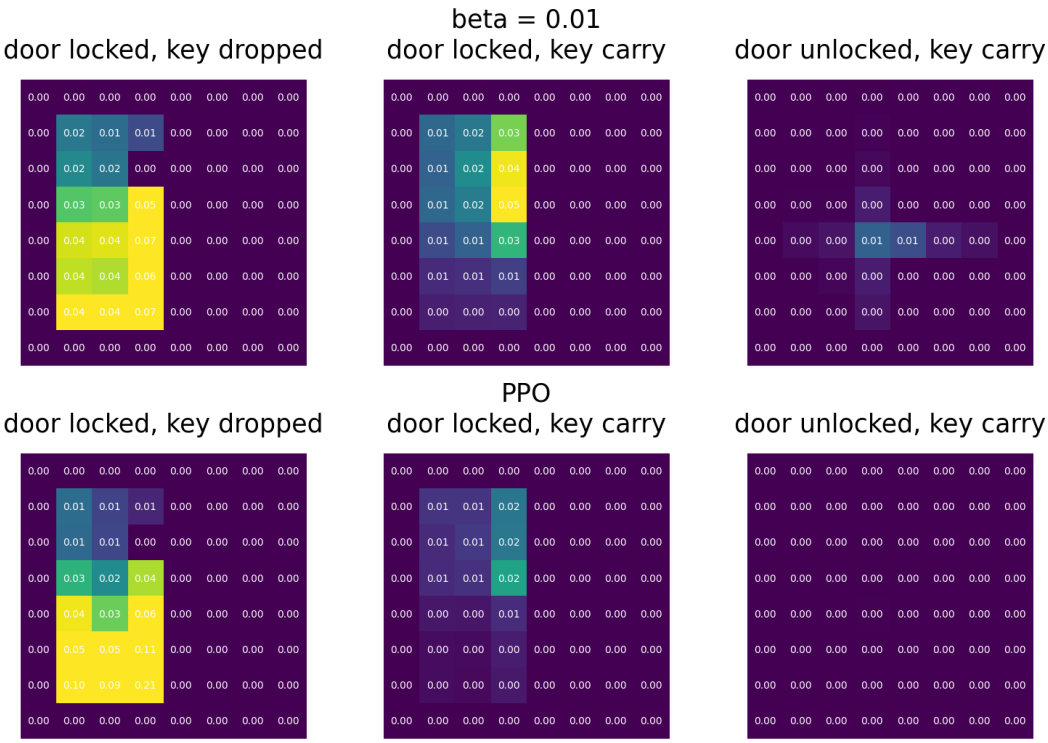


Figure 13: State distribution for modified DoorKey (100 steps) without reward after 200k steps of training PPO

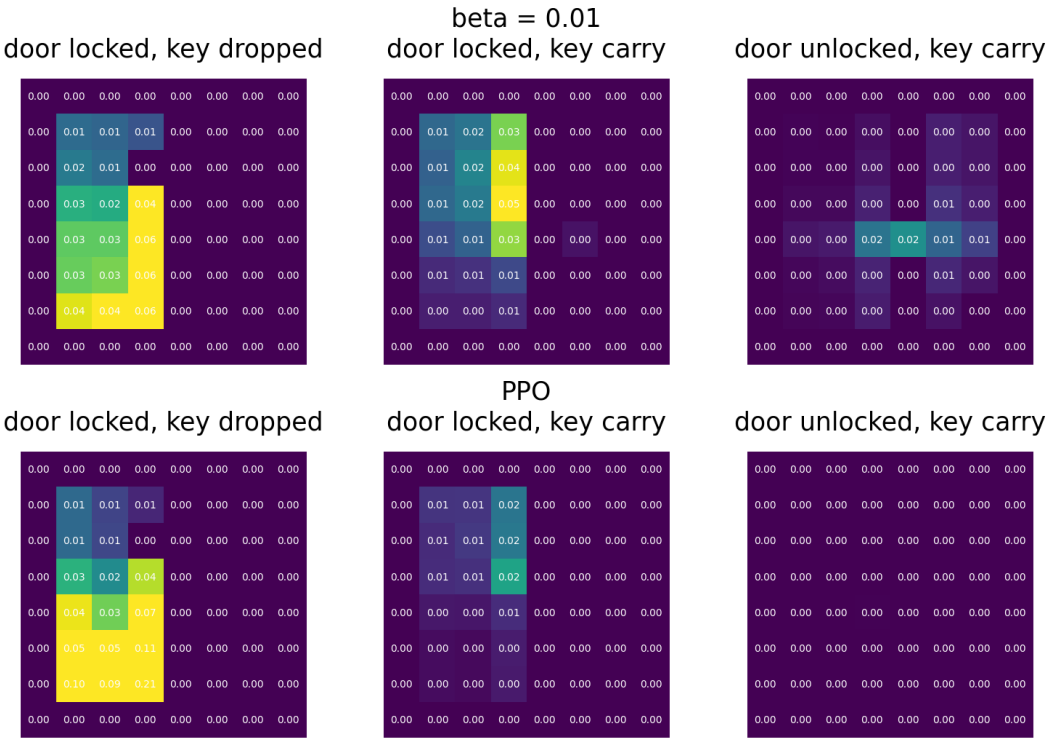


Figure 14: State distribution for modified DoorKey (100 steps) without reward after 500k steps of training PPO

Table 2: Hyperparameters for MiniGrid DoorKey

Hyperparameter	**Value**
PPO mini batch size	64
PPO iterations	10
PPO clipping factor (ϵ)	0.2
PPO value coefficient (c_1)	0.5
PPO entropy coefficient (c_2)	0.003
Discount factor (γ)	0.99
VQ-VAE epochs	8
VQ-VAE embedding vectors (k)	256

determined, the agent moves with a step size of 0.1 in the chosen direction (-0.1 for left and +0.1 for right). Additionally, Gaussian noise with zero-mean and variance of 0.01 is added to the movement. Choosing the left action moves the agent deterministically to the left, whereas choosing the right action moves the agent stochastically to the right, left, or keeps it in place. The left region of the river is defined as $s \in [0, 0.05]$, where selecting the left action yields a reward of $r = 5$. The right region of the river is defined as $s \in [0.95, 1]$, where the agent receives a high reward of 10,000 if it chooses the right action and remains in the same region. The problem is framed as a continuing problem with $\gamma = 0.99$, where the optimal policy is to always choose the right action, while the suboptimal policy results in getting stuck on the left side to collect the distractor reward. The full dynamics of the environment are shown in Figure 15.

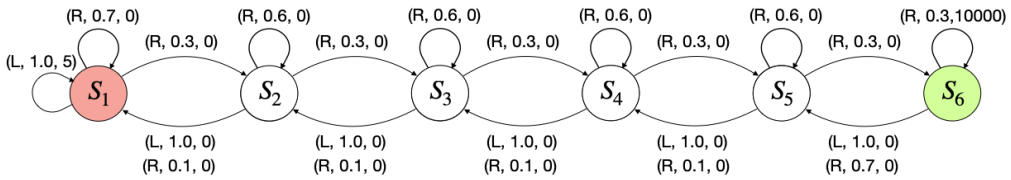


Figure 15: Riverswim environment. Agent starts at S_1 and each tuple is (Action, probability, reward).

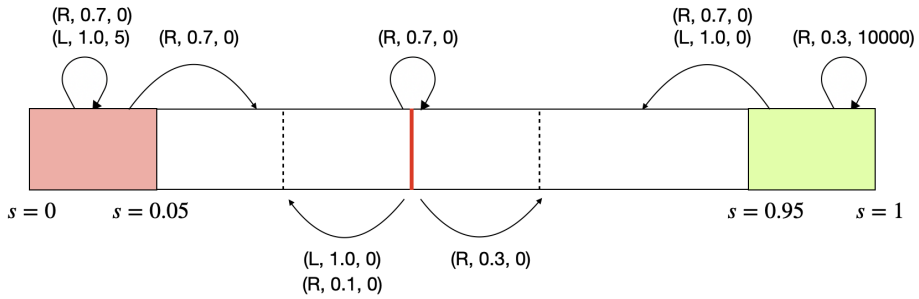


Figure 16: Continuous Riverswim environment. The low-reward region, $state \in [0.00, 0.05]$, where the agent receives $r = 5$ by moving left and the high-reward region, $s = [0.95, 1.00]$ where the agent receives $r = 10,000$ by moving right

S3.2 Mountain Car

Mountain Car (Moore, 1990) is a classic control environment consisting of a car positioned between two hills. The goal is to reach the top of the right hill. To achieve this, the car must build momentum by oscillating between the hills. Mountain Car has a continuous state space, defined by the car's position, $s_1 \in [-01.2, 0.6]$, and velocity, $s_2 \in [-0.07, 0.07]$. The action space is $a \in \{$accelerate left, accelerate right, don't accelerate$\}$. To encourage faster solutions, the agent receives a reward of -1 at each step. The episode terminates under one of two conditions: (1) the agent reaches the top of the hill ($s_1 \geq 0.45$), or (2) the episode length reaches 200 steps. This setup makes Mountain Car particularly challenging, as it is a sparse reward environment; therefore, the agent must explore effectively to reach the top of the hill. Additionally, the short episode length provides a limited window for the agent to explore efficiently.