Random Latent Exploration for Deep Reinforcement Learning

Srinath Mahankali [1] **Zhang-Wei Hong** [1] **Ayush Sekhari** [1] **Alexander Rakhlin** [1] **Pulkit Agrawal** [1]

Abstract

The ability to efficiently explore high-dimensional state spaces is essential for the practical success of deep Reinforcement Learning (RL). This paper introduces a new exploration technique called Random Latent Exploration (RLE), that combines the strengths of exploration bonuses and randomized value functions (two popular approaches for effective exploration in deep RL). RLE leverages the idea of perturbing rewards by adding structured random rewards to the original task rewards in certain (random) states of the environment, to encourage the agent to explore the environment during training. RLE is straightforward to implement and performs well in practice. To demonstrate the practical effectiveness of RLE, we evaluate it on the challenging ATARI and ISAACGYM benchmarks and show that RLE exhibits higher overall scores across all the tasks than other approaches, including action-noise and randomized value function exploration.

1. Introduction

Exploration is a central problem in Reinforcement Learning (RL) (Sutton & Barto, 2018), which is even more challenging in sparse-reward scenarios where the learning signal is rarely available. The essence of exploration lies in uncovering rewards that surpass those obtained from the current policy. Typical exploration approaches use action noise (e.g., ϵ-greedy, Boltzmann sampling, etc.) to search for useful reward signals. However, action noise alone cannot perform so-called *deep exploration* (Osband et al., 2016a) that requires the agent to discover rewards far away from the initial states.

Action noise fails to enable deep exploration because it produces dithering actions (i.e., moving back and forth around the initial state), which limit exploration to areas close to the initial states. There are currently two popular families of strategies for deep exploration: (i) adding exploration bonus to the reward (Pathak et al., 2017; 2019; Bellemare et al., 2016; Pritzel et al., 2017; Burda et al., 2019) and (ii) utilizing randomized value functions (Osband et al., 2013; 2016a;b; Fortunato et al., 2017; Ishfaq et al., 2023; 2021) for exploration. Based on the guidance of *Optimism in the Face of Uncertainty* (Lattimore & Szepesvári, 2020), exploration bonuses are purposed to estimate the novelty of states and encourage the policy to visit new states. As states distant from initial ones are seldom visited and hence novel, these bonuses facilitate deeper exploration. However, it necessitates training an additional deep neural network and fails to improve the action noise exploration method over the majority of tasks in both discrete (Chen et al., 2022) and continuous (Schwarke et al., 2023) control tasks, which limits the widespread adoption of exploration bonuses as a default exploration strategy in deep RL (Chen et al., 2022).

On the other hand, randomized value function methods are grounded on the framework of Thompson sampling (Russo et al., 2018), and are simpler to implement, as they do not necessitate estimating state novelty. These methods roll out trajectories using policies sampled from a distribution over learned value functions. As each trajectory is rolled out by the same policy, there are no action noises. Hence, those trajectories are temporally consistent (Dabney et al., 2020) and can move the agent further from initial states, thereby enabling exploration. However, neither bonus-based nor randomized value function methods improve over action noise exploration significantly (Chen et al., 2022).

We conjecture that current randomized value function approaches fail to substantially improve action noise based exploration strategies (Schulman et al., 2017; Chen et al., 2022; Taïga et al., 2019) because practical implementations of randomized value function approaches fail to represent a diverse enough value function distribution. If the value function distribution is not rich enough, the policies induced from them will be similar and hence won't be able to cover diverse states in the environments. These methods are thus ineffective at discovering states associated with rewards. The practical implementations either train an ensemble of value functions (Osband et al., 2016a) to approximate the

Project website: https://srinathm1359.github.io/random-latent-exploration

[1]Massachusetts Institute of Technology. Correspondence to: Srinath Mahankali <srinathm@mit.edu>.

Proceedings of the 41^{st} International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

distribution of value functions or perturb the parameters (i.e., neural network weights) of the learned value functions (Fortunato et al., 2017; Plappert et al., 2017). Unfortunately, ensemble-based approaches (Osband et al., 2016a) require a large ensemble to capture diverse value functions, which is memory-intensive and infeasible in practice. While parameter noise methods (Fortunato et al., 2017; Plappert et al., 2017) need less memory, Hong et al. (2018) and our experiments (Section 4.1) have shown that these methods fail to produce diverse trajectories.

In this paper, we aim to provide an exploration strategy that is easy to implement and effective in practice across a variety of domains spanning both discrete and continuous control tasks. To this end, we bridge the best of both worlds in exploration bonus and randomized value function approaches: we train the policy with perturbed rewards that are obtained by adding random rewards to the original reward (i.e., task reward received from the environment). Altering rewards has been shown to be effective at producing diverse trajectories in prior works of skill discovery (Eysenbach et al., 2018). As our experiments in Section 4.1 show, introducing randomness to rewards influences the policy to produce diverse behaviors, leading to diverse trajectories. These randomized rewards act as a *"randomized bonus"* for the policy. Furthermore, because they can be randomly sampled in a simple way, implementing randomized rewards is straightforward in practice. Meanwhile, the specific method by which novelty is estimated is crucial for bonus-based exploration (Houthooft et al., 2016; Bellemare et al., 2016; Ostrovski et al., 2017; Pathak et al., 2019), therefore making it more complex to implement.

To show the effectiveness of our approach, we experiment with Random Latent Exploration (RLE), our exploration strategy, in ATARI—a popular discrete action space deep RL benchmark (Bellemare et al., 2013), and ISAACGYM—a popular continuous control deep RL benchmark (Makoviychuk et al., 2021), each consisting of many different tasks with varying degrees of exploration difficulty. We implement our method on top of the popular RL algorithm, Proximal Policy Optimization (PPO) (Schulman et al., 2017) and compare it with PPO in other exploration strategies. Our experimental results demonstrate that RLE improves over standard PPO in ATARI and ISAACGYM. Furthermore, RLE also exhibits a higher aggregated score across all tasks in ATARI than other exploration methods, including RND and randomized value function strategies (Fortunato et al., 2017). Importantly, these improvements were obtained by simply adding RLE on the top of the base PPO implementation (with minimal change to hyperparameters), thus highlighting the generality of our approach as a plug-in utility.

2. Preliminaries

Reinforcement Learning (RL). RL is a popular paradigm for solving sequential decision-making problems (Sutton & Barto, 2018) where an agent operates in an unknown environment (Sutton & Barto, 2018) and aims to improve its performance through repeated interactions with the environment. At each round of interaction, the agent starts from an initial state s_0 of the environment and collects a trajectory. At each timestep t within that trajectory, the agent perceives the state s_t, takes an action $a_t \sim \pi(.|s_t)$ with its policy π, receives a task reward $r_t = r(s_t, a_t)$, and transitions to a next state s_{t+1} until reaching terminal states, after which a new trajectory is initialized from s_0 and the above repeats. The goal of the agent is to learn a policy π that maximizes expected return $\mathbb{E}_\pi\Big[\sum_{t=0}^{\infty}\gamma^t r(s_t, a_t)\Big]$ in a trajectory. A straightforward approach is to estimate the expected return of a policy by rolling out trajectories $(s_0, a_0, s_1, \cdots,)$ through Monte Carlo sampling (Konda & Tsitsiklis, 1999), and then optimizing this to find the optimal policy, but unfortunately, the corresponding estimates are of high variance and thus often require a huge number of data. Thus, in practice, various RL algorithms learn a value function (or value network) V^π from the interaction that approximates

$$V^\pi(s_0) \approx \mathbb{E}_\pi\Big[\sum_{t=0}^{\infty}\gamma^t r(s_t, a_t)\Big], \tag{1}$$

and train the policy π to maximize the value $V^\pi(s_0)$ (e.g. using policy gradient).

Exploration. As the reward may be delayed and not immediately presented to the agent, the agent may need to take many actions and visit a sequence of states without rewards before it receives any learning signal (reward). As such, taking greedy action a_t at each step that maximizes immediate reward $r(s_t, a_t)$ does not necessarily lead to a high return. Thus, RL algorithms require "exploring" states and actions that may lead to low immediate rewards but could potentially end up with high return in the long run. We refer to this process as *exploration* throughout this paper.

3. Our Method: Random Latent Exploration

Problem statement. Our goal in this paper is to create an exploration strategy that improves the standard action noise exploration method overall on the majority of tasks and domains (both discrete and continuous control) and is yet easy to implement. Toward that end, we provide a simple exploration method that combines the best of both worlds and bridges the approach of using reward bonuses (Bellemare et al., 2016; Burda et al., 2019) with randomization for exploration (Osband et al., 2016a;b).

Challenge: Ensuring diversity. The key challenge in randomized exploration methods lies in their ability to generate diverse behaviors within the environment. Specifically, if

the value functions and policies sampled at random bear too much similarity, they will yield nearly identical trajectories $(s_0, a_0, r_0, s_1, \cdots)$, consequently limiting exploration to a narrow region of the environment. The primary technical challenge, then, is devising a method that ensures such diversity in the generated trajectories.

Approach: Randomized rewards. To overcome the above challenge, we propose enhancing trajectory diversity by altering the agent's rewards, inspired by skill discovery methods (Eysenbach et al., 2018) that use varied reward functions to develop diverse skills resulting in different state visitation distribution. In particular, in every episode of training, we randomly perturb the given task reward by adding a randomized reward to certain (random) parts of the state space. Then, we train policies to maximize the composite of randomized and task rewards. The key idea is that in every round of interaction, the random rewards added to certain random parts of the state space will incentivize the agent to visit those areas, and if these random areas are diverse enough, we will get diverse behaviors in the environment during training—thus incentivizing exploration. However, since the random rewards are repeatedly resampled and thus keep on changing during training to ensure stable and effective learning, both policies and value functions must be aware of the specific random reward function in use; otherwise, the changing reward functions will comprise a partially observable MDP (Kaelbling et al., 1998). We take inspiration from the Universal Value Function Approximator (UVFA) (Schaul et al., 2015), which trains networks based on varying goals. We adopt their approach by equating their goals to different reward functions, thus making the policy π and the value function V^π condition on the sampled reward function. This ensures that the random rewards no longer appear as noises to the policy. The remaining questions are:

- How to implement the randomized reward functions?
- How to make the policy and the value function condition on the sampled reward functions?

We outline our implementation of the above idea in the next section and defer full implementation details to Appendix B.

3.1. Algorithmic Implementation

Randomized reward functions. An effective approach to implementing randomized reward functions necessitates adherence to two fundamental principles. First, the randomized rewards require correlating with states; otherwise, they would appear as white noises, which does not help exploration as shown by Fortunato et al. (2017). Second, the function must be factorized to allow both the policy and the value function to condition the factors of the randomized reward function. This avoids the perception of the reward as partially observable by the policy. To fulfill these criteria, we implement the randomized reward function (denoted as F) as the dot product of the state feature and a randomly chosen latent vector:

$$F(s, \boldsymbol{z}) = \phi(s) \cdot \boldsymbol{z}, \tag{2}$$

where $\phi : \mathcal{S} \to \mathbb{R}^d$ is a feature extractor that transforms a state into a d-dimensional vector, and $\boldsymbol{z} \in \mathbb{R}^d$ represents a latent vector. Randomized rewards for each state are generated by sampling $\boldsymbol{z}$ from a given distribution $P_{\boldsymbol{z}}$, and then setting rewards as $F(s, \boldsymbol{z})$.

Latent conditioned policy and value network. Recall that the policy and the value function have to be aware of the state and the random variable that factorizes the randomized reward function F. To achieve this, we augment the input to the policy and the value functions with the latent vector $\boldsymbol{z}$. The resulting policy and the value networks are $\pi(.|s, \boldsymbol{z})$ and $V^\pi(s, \boldsymbol{z})$. We train the latent-conditioned value network to approximate the expected sum of the original reward and the randomized rewards as below

$$V^\pi(s, \boldsymbol{z}) \approx \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t (R(s_t, a_t) + F(s_{t+1}, \boldsymbol{z})) \right], \tag{3}$$

and train the latent-conditioned policy π to maximize $V^\pi(s, \boldsymbol{z})$ at every state s and latent vector $\boldsymbol{z}$. Both value and policy networks can be trained with any off-the-shelf RL algorithms, e.g. PPO (Schulman et al., 2017), DQN (Mnih et al., 2015), A3C (Mnih et al., 2016), SAC (Haarnoja et al., 2018).

Latent vector sampling. To randomize the latent vector $\boldsymbol{z}$, rather than resampling it at each timestep, which could cause dithering actions due to fast-changing latent vectors fed to the policy, it is resampled at the start of each trajectory. This ensures each trajectory is rolled out under the same policy and latent vector $\boldsymbol{z}$, maintaining temporal consistency crucial for deep exploration, as indicated by prior studies (Osband et al., 2016a; Fortunato et al., 2017). The distribution of $\boldsymbol{z}$ will be elaborated in Section 4.

As we train the policy conditioned on the latent factor of the randomly sampled reward function, we term our method as *Random Latent Exploration (RLE)*. We outline the algorithm in Algorithm 1 and present the detailed version in Algorithm 2. Note that at line 6 in Algorithm 1, we compute the randomized reward at the next state s_{t+1} since the next state reflects the credit of taking action a_t and state s_t. This choice is also common in prior works that compute exploration bonuses at each step (Burda et al., 2019).

Algorithm 1 Random Latent Exploration (RLE)

Input: Latent distribution $P_{\boldsymbol{z}}$
repeat
  Sample a fresh latent vector: $\boldsymbol{z} \sim P_{\boldsymbol{z}}$
  for $t = 0, \ldots, T$ **do**
    Take action $a_t \sim \pi(.|s_t, \boldsymbol{z})$ and transition to s_{t+1}
    Receive reward: $r_t = R(s_t, a_t) + F(s_{t+1}, \boldsymbol{z})$
  end for
  Update policy network π and value network V^π with the collected trajectory $(\boldsymbol{z}, s_0, a_0, r_0, s_1, \cdots, s_T)$
until convergence

4. Experiments

We aim to show that our RLE improves over the action noise exploration method typically used in many RL algorithms (Schulman et al., 2017; Mnih et al., 2015) on the majority of tasks in both discrete and continuous control domains. Throughout the experiments, we train the agent for each task using PPO (Schulman et al., 2017) since it is a popular RL algorithm used in both discrete and continuous control tasks. Standard PPO implementation (Schulman et al., 2017) explores by sampling actions from the policy (i.e., a Boltzmann distribution over actions) learned.

We also compare RLE with the following exploration strategies as baselines:

- NOISYNET (Fortunato et al., 2017): We chose it to be the representative baseline from the family of randomized value function exploration strategies (Osband et al., 2016a; Fortunato et al., 2017; Plappert et al., 2017) because it has been used in prior works on benchmarking exploration strategies (Chen et al., 2022; Taïga et al., 2019).
- RND (Burda et al., 2019): We choose RND to be the representative baseline from the family of bonus-based exploration methods since it shows considerable improvements over action noises and randomized value function approaches in hard-exploration tasks in ATARI.

4.1. Illustrative Experiments on FOURROOM

To test whether our method can perform deep exploration that is essential in many tasks requiring exploration, we start by running toy experiments on the the FOURROOM environment (Sutton et al., 1999). We explain the setting of FOURROOM environment and the results as follows.

Setup. Figure 1 illustrates FOURROOM environment with 50×50 states. The environment consists of four rooms separated by solid walls (which the agent can't cross) and connected with small openings of a single state each (which the agent needs to go through to travel across rooms). The agent perceives the (x, y) coordinates as the state input, and can take an action to move left, right, up, and down (if not interrupted by a wall). At the beginning of each trajectory, the agent always starts from the top-right corner of the room (denoted by the letter "S"). In this study, we always give zero rewards to the agent since we are interested in understanding and comparing how each exploration strategy behaves in the presence of sparse rewards, or even zero everywhere. This is also known as reward-free exploration.

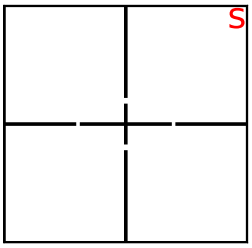


Figure 1. FOURROOM environment. The agent starts at the top-right state (denoted by red 'S') and can move left, right, up, and down. The black bars denote walls that block the agent's movement.

We compared the agents trained with different exploration strategies: PPO, NOISYNET, RND, and RLE. The corresponding policy in each of these exploration strategies is trained with Proximal Policy Optimization (PPO) algorithm (Schulman et al., 2017) for 2.5 million timesteps. For RLE, the feature extractor defined in Equation (2) is set to be a randomly initialized neural network with one hidden layer, the output layer of which has the same dimension as $\boldsymbol{z} \sim P_{\boldsymbol{z}}$. Further implementation details are available in Appendix B.2. We remark that, because of the walls, the FOURROOM environment requires deep exploration to go to states distant from the initial states.

How does the latent vector influence the generated trajectories? The exploration of RLE is driven by sampling the latent vector $\boldsymbol{z}$ to change the behavior of the policy network π and the objective of the value network V^π. One may wonder what is the impact of $\boldsymbol{z}$ on the induced policy's behaviors. To evaluate this, we sample different latent vector $\boldsymbol{z} \sim P_z$ and rollout trajectories with the policy conditioning on those $\boldsymbol{z}$, plotting each trajectory in a different color in Figure 2. For this plot, we chose the checkpoint of the policy network stored in the middle of training (i.e., 1.5 million timesteps) since we want to study the behaviors of the policy network before it converges. Figure 2 demonstrates that the trajectories generated by different latent vectors $\boldsymbol{z}$ can span across all four rooms. This shows that altering the latent vector $\boldsymbol{z}$ can produce diverse trajectories and that latent vectors can influence the behaviors of the policy network. We provide the rollouts of trajectories for random seeds for RLE, PPO, RND and NOISYNET in Appendix C.

Explaining the observed trajectory diversity. To explain why the trajectories generated by an RLE policy are diverse,

We also perform experiments on FOURROOM with an sparse task reward of 1 at the bottom-left corner. The results and visualization of visitation counts are deferred to Appendix C.

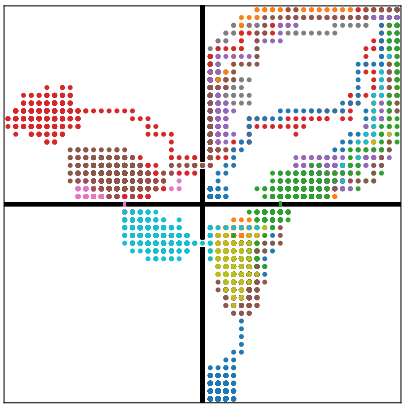


Figure 2. Rollout of multiple trajectories from a policy trained with RLE in the middle of the training (1.5 million timesteps), where each color denotes a distinct trajectory. As the figure demonstrates, changing the latent vector $\boldsymbol{z}$ in RLE leads to diverse trajectories across all four rooms.

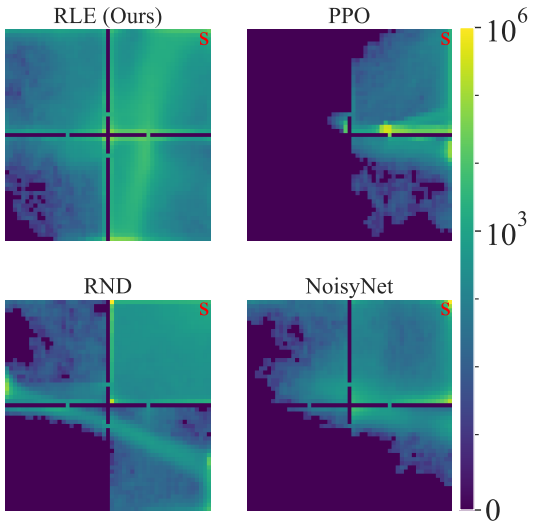


Figure 3. State visitation counts of all the methods after training for 2.5M timesteps without any task reward (reward-free exploration). The start location is represented by the red 'S' at the top right.

we plot several of the reward functions induced by sampling different $\boldsymbol{z}$ in Figure 11. These plots demonstrate the diversity of the random rewards, each of which can guide the policy to a different part of the state space.

State Visitations in the environment. We plot the state visitation counts of each exploration strategies and present the results in Figure 3. The results shows that PPO's state visitation centers around the initial room (i.e, top-right), indicating that action noises are not able to bring the agent far away from the initial state. In contrast, we see that RLE, RND, and NOISYNET are all able to frequently reach the rooms beyond the initial room, with RLE visitation count spread out across the four rooms. This suggests that RLE is capable of doing deep exploration similar to prior deep exploration algorithms for this environment.

4.2. Benchmarking Results on ATARI

Having performed illustrative experiments on the FOURROOM toy environment, we now evaluate our method on more realistic and challenging tasks. We aim to show that RLE can improve PPO's overall performance on most tasks.

Setup. We evaluate our method in the well-known ATARI benchmark (Bellemare et al., 2013). Following the common practice in ATARI (Mnih et al., 2015), the agent perceives a stack of the most recent four 84×84 grayscale frames as inputs and takes discrete actions available in the gamepad of ATARI (see Bellemare et al. (2013) for further environment details). For RLE, we chose the feature learned by the value network followed by a randomly initialized linear layer as ϕ (used in Equation 2) and set the dimension of the latent vector $\boldsymbol{z}$ as 8. Note that the randomly initialized linear layer is frozen after initialization. We use the standard PPO hyperparameters provided in (Burda et al., 2019).

The hyperparameters and implementation details of all the algorithms and PPO are deferred to Appendix B. For each ATARI game (i.e., environment), we train each agent with 5 different random seeds for 40 million frames, as prior work (Chen et al., 2022; Bellemare et al., 2016) suggested. However, we trained MONTEZUMA'S REVENGE for 200 million frames since its exploration difficulty is much harder than other ATARI games (Burda et al., 2019).

Does RLE improve the overall performance? We answer this question by calculating the interquartile mean (IQM) (Agarwal et al., 2021) and its 95% confidence interval, which was estimated using the bootstrapping method (DiCiccio & Efron, 1996) on the aggregated human-normalized scores from 57 games. Unlike empirical mean scores, IQM mitigates the influence of outliers on the aggregated metric. Figure 4 demonstrates that RLE achieves a higher IQM human-normalized score compared to all baselines, indicating that RLE enhances performance over other exploration strategies in the majority of ATARI tasks. Besides the aggregate results, we present the learning curves for all methods across the 57 ATARI games in Figure 19. Additionally, the final mean score of each method across five seeds for each ATARI game is provided in Table 6 in the appendix.

Does RLE improve over the baselines consistently? In addition to the margin of performance improvement on the aggregated score across all games, we demonstrate that RLE results in performance improvement over the baselines with high probability. Following the evaluation protocol suggested in Agarwal et al. (2021), we measure the probability of improvement (POI) between algorithms and

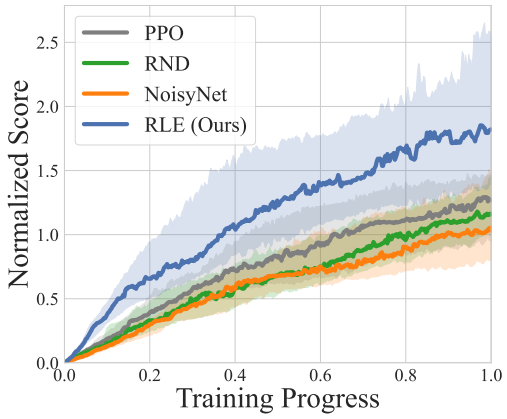
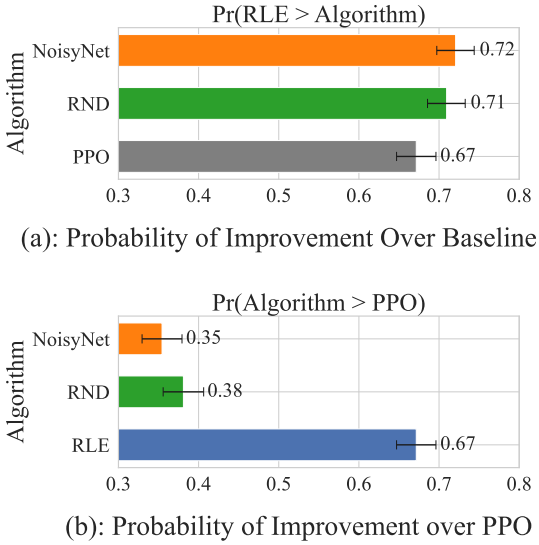


Figure 4. Aggregated human normalized score across all 57 ATARI games. RLE exhibits a higher interquartile mean (IQM) of normalized score than PPO across 57 ATARI games, showing that RLE improves over PPO in the majority of tasks.



(a): Probability of Improvement Over Baseline

(b): Probability of Improvement over PPO

Figure 5. **(a)** Probability of improvement of our method, RLE, over the baselines NOISYNET, RND and PPO across all 57 ATARI games. The lower confidence bound of RLE's POI over the other algorithms are all greater than 0.5. This means that RLE statistically improves the other algorithms (Agarwal et al., 2021). **(b)** Probability of improvement of RLE, RND, and NOISYNET over PPO across all 57 ATARI games. POI over PPO of both NOISYNET and RND are below 0.5, implying that neither NOISYNET nor RND statistically improve over PPO overall across 57 ATARI games.

their 95% confidence intervals, estimated using the bootstrapping method (DiCiccio & Efron, 1996), and present the results in Figure 5. Figure 5(a) shows that the lower confidence bound of POI for RLE over each algorithm is above 0.5, indicating that RLE statistically outperforms the other baselines (Agarwal et al., 2021). This means that for a randomly chosen task in ATARI, running RLE is highly likely to yield a higher score than the other baselines, implying that RLE's performance improvements are consistent and not limited to a few games. Conversely, Figure 5(b) reveals that the POI over PPO for both NOISYNET and RND is far below 0.5, suggesting that NOISYNET and RND do not consistently improve over PPO despite having better performance in a few games (see Figure 19).

4.3. Benchmarking Results on ISAAC GYM

To demonstrate that RLE can improve upon PPO in both discrete and continuous control tasks, we also conducted experiments in ISAACGYM (Makoviychuk et al., 2021), a benchmark suite containing numerous continuous control tasks. We implemented RLE on top of PPO and trained it with standard PPO hyperparameters in ISAACGYM provided in `CleanRL`, with implementation details provided in Appendix B.4. We selected both manipulation and locomotion tasks for evaluation and presented the learning curves in Figure 7, with learning curves on all environments presented in Figure 23. As the return is positive for PPO in each environment, we normalize runs by dividing by the mean score of PPO in that environment (i.e., normalize PPO to have a score of 1). For further details of this metric, see Appendix B.4.2.

Does RLE improve over PPO in continuous control? The results show that RLE achieves a higher average return than PPO in most tasks, with particularly large performance gains in ALLEGROHAND and SHADOWHAND, indicating that RLE improves upon PPO in continuous control tasks. In ALLEGROHAND and SHADOWHAND, the objective is to control a anthropomorphic hand to reorient objects to a target pose. These tasks require more exploration than other continuous control tasks since it takes many steps to achieve the target pose. Additionally, in CARTPOLE, PPO performance degrades abruptly in the middle of training, while RLE maintains high performance throughout, suggesting that RLE prevents the learning process from collapsing during training.

Also, Table 1 shows that RLE achieves a lower confidence bound of POI above 0.5, indicating that RLE improves over PPO in a statistically high probability.

Table 1. Probability of Improvement (POI) of RLE (Ours) over PPO (higher is better) in ISAACGYM

	RLE over PPO
POI	0.66
95% CI	[0.60, 0.72]

Does RLE achieve higher overall performance than PPO? To study the overall performance, we also measure the IQM of the normalized return of RLE and PPO. The return is normalized

so that the mean score of PPO in each environment is 1. Figure 6 presents the IQM of normalized return, showing that RLE achieves higher IQM over PPO and thereby indicating RLE improves the overall performance on most tasks in IsaacGym.

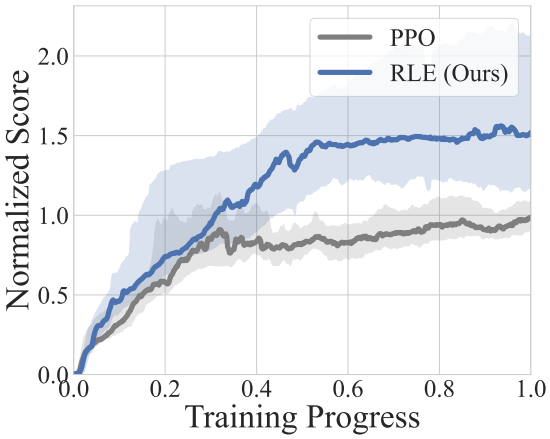


Figure 6. Aggregated normalized score across all 9 IsaacGym tasks that we consider. RLE achieves a higher interquartile mean of normalized score compared to PPO, indicating that it can improve over PPO in continuous control domains as well.

4.4. Ablation Studies

We ran various ablation studies on Atari and IsaacGym environments to explore how different hyperparameters and design choices affect RLE performance.

Latent vector distribution. We investigated the impact of different latent vector distributions on RLE's performance. Our study involves training RLE with various distributions, including $\mathrm{Uniform}([-0.5, 0.5]^d)$ and isotropic normal $\mathcal{N}(\mathbf{0}, \mathbf{I}_d)$ distributions, within a d-dimensional space where $d = 8$. The detailed implementation is described in Appendix B. The results presented in Figure 8 indicate that RLE performs better than PPO across different latent vector distributions. This suggests that RLE's efficacy is not significantly affected by the choice of latent vector distribution.

Latent vector dimension. This study explores how robust is RLE to different choices of the dimension d of the latent vector $\mathbf{z}$. We trained RLE for $d \in \{2, 8, 32, 128\}$, where $d = 8$ is the dimension used in the results presented in Section 4.2. The outcomes, depicted in Figure 9, demonstrate that RLE is capable of surpassing PPO across all tested dimensions. Although there are slight performance variations between different d values, these differences are subtle, suggesting that RLE's performance is insensitive to the choice of latent vector dimension d.

Latent vector conditioning. In Section 3.1, we emphasized the necessity for the policy to be conditioned on the latent vector to prevent randomized rewards $F(s, \boldsymbol{z})$ from being perceived as noise by both the policy and the value network. This design choice's importance is underscored by comparing RLE models with and without latent-conditioned policy and value networks, as shown in Figure 10. RLE without latent vector conditioning exhibited a performance drop in the Venture task, a hard-exploration game with sparse rewards. We hypothesize that the absence of latent vector conditioning results in limited behavioral variability in the policy network, as its outputs remain unchanged by different latent vector samples. This limitation likely leads to failures in challenging exploration tasks that necessitate a broader diversity in trajectory generation.

Features vectors obtained from random neural network. In the Atari experiments described in Section 4.2, we used a slow-moving estimate of the CNN features (see Appendix B.3.1) learned by the value network to compute RLE rewards. This choice of features slightly contributes to improved performance. Figure 21 presents the IQM of the normalized score and the POI of RLE over PPO with and without using the CNN features learned by the value network. The results demonstrate that incorporating value network features leads to a higher POI, while not significantly affecting the IQM. We plot the learning curve in each game for both variants in Figure 22, finding that while performance is broadly similar, there are a few games where the two variants have large differences in performance.

Choices of network architecture for the random reward network. Since RLE relies on neural networks to extract features for computing random rewards $F(s, \boldsymbol{z})$, it's important to examine how the choice of network architecture affects performance. We investigated the impact of different network architectures for extracting features $\phi(s)$ on the computation of RLE rewards $F(s, \boldsymbol{z})$ in IsaacGym (Makoviychuk et al., 2021), with the IQM of the normalized score in Figure 24 and POI over PPO shown in Figure 25. In our original IsaacGym experiments, we used the value network's architecture for RLE. In this ablation study, we tested a shallower neural network architecture. The results indicate that RLE with a shallower network still performs well, suggesting that RLE is not highly sensitive to the choice of network architecture.

White noise random rewards. In Section 3.1, we emphasized the importance of ensuring that RLE's random reward function F is correlated with states. If the random rewards are not state-dependent, they will act as white noise and will not enhance performance. In this ablation study, we empirically investigate the significance of making random rewards dependent on states. We compare the state-dependent RLE reward $F(s, \boldsymbol{z})$ with white noise rewards sampled from a normal distribution with zero mean and unit variance. This

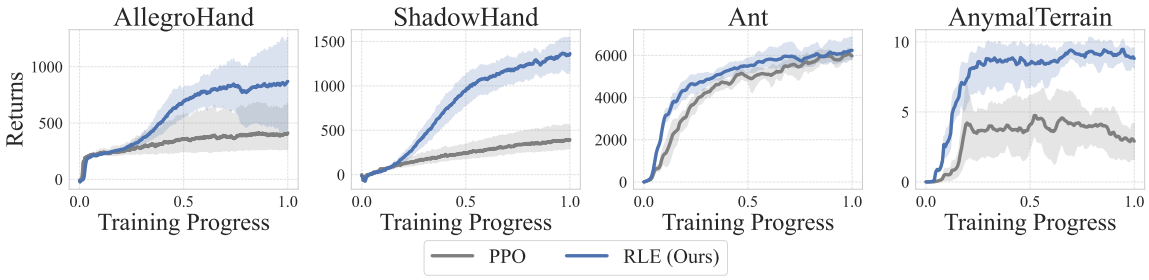


Figure 7. Comparison of learning curves between RLE and standard PPO. RLE achieves higher return than the standard PPO in the majority of the tasks, especially in tasks like ALLEGROHAND and SHADOWHAND that require more exploration. This means RLE improves over PPO in continuous control domain as well.

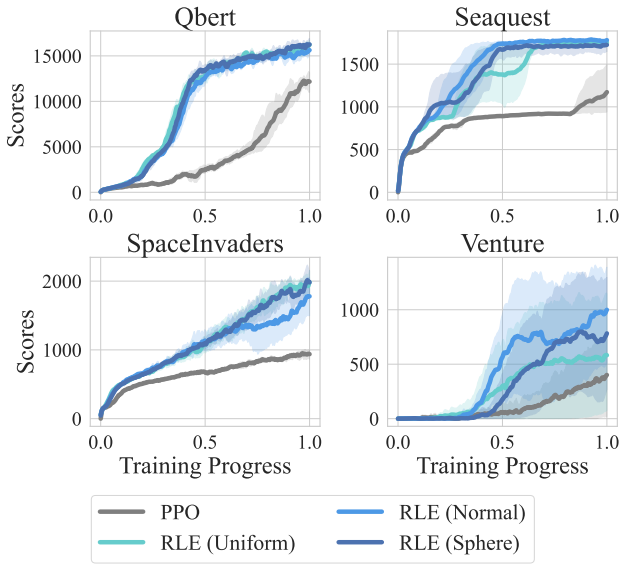


Figure 8. Learning curves of RLE with varying latent vector distribution $P_{\boldsymbol{z}}$ (see Section 3.1), where RLE (Sphere) is the one used in Section 4.2. The figure shows that RLE with the three distributions can all outperform PPO. This shows that RLE is not sensitive to the choice of latent vector distribution.

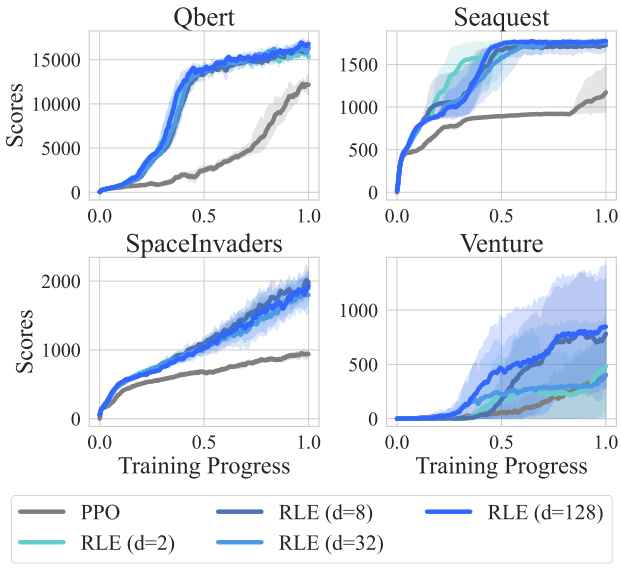


Figure 9. Learning curves of RLE with varying latent vector dimension d (see Section 3.1), where RLE ($d = 8$) is the one used in Section 4.2. The figure shows that RLE can outperform PPO in all the four chosen dimension. This shows that RLE is not sensitive to the choice of latent vector dimension.

study was conducted in ISAACGYM environments, with the results presented in Figure 26. The results demonstrate that white noise rewards significantly degrade performance, indicating that RLE rewards are not merely white noise.

5. Related Works

Random reward prediction was used as an auxiliary task (Jaderberg et al., 2016) for improving representation learning in prior works (Dabney et al., 2021; Lyle et al., 2021). A closely related work is Ramesh et al. (2022), which employs a random general value function (GVF) (Sutton et al., 2011) for exploration by initializing a random reward function and using an ensemble of networks to predict policy-based random reward sums. The difference in prediction and the Monte Carlo estimate of random rewards, multiplied by prediction variance, enhances the agent's reward. Our work presents a distinct approach from Ramesh et al. (2022) both in terms of motivation and implementation. Contrary to Ramesh et al. (2022), which aligns with previous studies (Burda et al., 2019; Pathak et al., 2017) by employing prediction errors as exploration bonuses, our RLE algorithm directly trains the policy with random rewards, demonstrating superior performance. This finding underscores that RLE provides a new angle to design exploration strategy beyond using prediction errors as exploration bonuses. Additionally, our RLE algorithm offers a more straightforward implementation by eliminating the need for ensemble train-

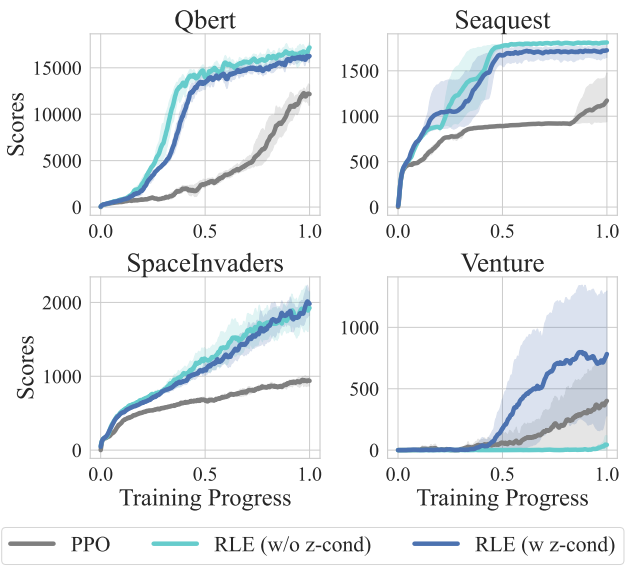


Figure 10. Learning curves of RLE with (w-cond) and without (w/o z-cond) latent vector condition in the policy and value networks (see Section 3.1). The figure displays that RLE without latent vector condition suffers performance drop in VENTURE, a hard-exploration task with sparse rewards.

ing and Monte Carlo return estimation of random rewards. The detailed discussion on the relevant literature can be found in Appendix A.

6. Discussion and Conclusion

In this paper, we proposed a new exploration method called RLE that is straightforward to implement in practice, and effective in challenging deep RL benchmarks like ATARI. We conclude with discussions and future work directions:

Simple plug-in for deep RL algorithms. RLE simply requires adding randomized rewards to the rewards received by the agent and augmenting the agent's input with additional latent variables that correlate to these randomized rewards. As a result, RLE is agnostic to the base RL algorithm and can be integrated with any RL algorithm. Given its simplicity, generality, and the overall performance improvement it provides, we recommend using RLE as the default exploration strategy in deep RL implementations.

Connection to posterior sampling. At a high level, while RLE seems similar to the posterior sampling-based approaches (Thompson, 1933; 1935; Russo et al., 2018) in the sense that both utilize randomization for exploration, there are important differences: Firstly, the two methods explore via different mechanisms. Posterior sampling randomizes over different models of the environment, whereas RLE perturbs the reward function using random rewards. Secondly, the sampling distribution P_z is fixed throughout learning in RLE, whereas the posterior distribution in posterior sampling changes with time and needs to be computed for every round (which is often challenging in practice). Thirdly, in posterior sampling, the posterior will eventually concentrate around the true model, and thus the algorithm will execute the optimal policy for the underlying environment. Whereas, in RLE, since the task rewards are constant throughout learning whereas random rewards change, in the later stage of the learning, the trained policy π_z should focus on optimizing just the task rewards, and we believe that the random rewards will simply act as a regularization.

Benefits from parallelization. Note that, by design, our algorithm samples independent $\boldsymbol{z}$ in every round and can thus benefit from parallelization by running the algorithm on multiple copies of the same environment (when possible, e.g. using a simulator). Since different $\boldsymbol{z}$ produce diverse trajectories (see Figure 2 or Figure 15 for illustrations), multiple parallel copies of the same agent will simply produce more diverse data which would accelerate exploration.

On the inductive bias of ϕ. RLE is modular as one can choose any feature extractor $\phi(s)$ e.g. Transformer networks (Vaswani et al., 2017), MLPs, or even nonparametric models such as kernels. In our ATARI experiments we use a CNN for ϕ, but it would be interesting to explore how other choices of ϕ affect the diversity of the induced reward functions, and hence the generated trajectories.

z-sampling. At every timestep in Algorithm 1 the latent variable $\boldsymbol{z}$ is sampled independently from the fixed distribution P_z which is chosen at initialization. However, it is also intuitive to expect that P_z should change as we learn more about the underlying environment. Looking forward it would be interesting to explore algorithms that change P_z while training, e.g. to sample more from the set of latent variables which have not been explored yet or for which the corresponding policies π_z have historically performed well in the given environment.

Other limitations. Currently, we limit our study to on-policy algorithms. Looking forward it would be interesting to extend RLE to off-policy algorithms such as DQN (Mnih et al., 2015) and SAC (Haarnoja et al., 2018); A practical way to do so would be to condition the Q-function on $\boldsymbol{z}$ in addition to its usual inputs. Separately, an important direction for future work is to explore the method in more continuous control and real-world robotics domains. While it is clear that our approach scales to high-dimensional state spaces in ATARI and continuous control tasks in ISAACGYM, it would also be interesting to see how it would generalize for real-world RL applications, e.g. in robotics.

Acknowledgement

The contributions of all authors are listed at the project website. We thank members of the Improbable AI Lab for helpful discussions and feedback. We are grateful to MIT Supercloud and the Lincoln Laboratory Supercomputing Center for providing HPC resources. This research was supported in part by Hyundai Motor Company, Quanta Computer Inc., MIT-IBM Watson AI Lab, an AWS MLRA research grant, ARO MURI under Grant Number W911NF-23-1-0277, ARO MURI under Grant Number W911NF-21-1-0328, and ONR MURI under Grant Number N00014-22-1-2740, NSF through award DMS-2031883, DOE through award DE-SC0022199, MIT UROP through the John Reed UROP Fund and the Paul E. Gray (1954) UROP Fund, and Simons Foundation. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Office or the United States Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes, notwithstanding any copyright notation herein.

Impact Statement

This paper presents work whose goal is to advance the field of Machine Learning. Our approach aims to accelerate RL in real-world domains, and depending on the domain to which it will be applied, we see many potential societal consequences of our work, however, none of them we feel must be specifically highlighted here.

References

Agarwal, A., Henaff, M., Kakade, S., and Sun, W. Pc-pg: Policy cover directed exploration for provable policy gradient learning. *Advances in neural information processing systems*, 33:13399–13412, 2020.

Agarwal, R., Schwarzer, M., Castro, P. S., Courville, A. C., and Bellemare, M. Deep reinforcement learning at the edge of the statistical precipice. *Advances in Neural Information Processing Systems*, 34, 2021.

Auer, P., Jaksch, T., and Ortner, R. Near-optimal regret bounds for reinforcement learning. *Advances in neural information processing systems*, 21, 2008.

Ayoub, A., Jia, Z., Szepesvari, C., Wang, M., and Yang, L. Model-based reinforcement learning with value-targeted regression. In *International Conference on Machine Learning*, pp. 463–474. PMLR, 2020.

Badia, A. P., Piot, B., Kapturowski, S., Sprechmann, P., Vitvitskyi, A., Guo, Z. D., and Blundell, C. Agent57: Outperforming the atari human benchmark. In *International Conference on Machine Learning*. PMLR, 2020.

Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. Unifying count-based exploration and intrinsic motivation. In *NIPS*, 2016.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

Burda, Y., Edwards, H., Storkey, A., and Klimov, O. Exploration by random network distillation. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=H1lJJnR5Ym.

Cesa-Bianchi, N., Gentile, C., Lugosi, G., and Neu, G. Boltzmann exploration done right. *Advances in neural information processing systems*, 30, 2017.

Chen, E., Hong, Z.-W., Pajarinen, J., and Agrawal, P. Redeeming intrinsic rewards via constrained optimization. *arXiv preprint arXiv:2211.07627*, 2022.

Dabney, W., Ostrovski, G., and Barreto, A. Temporally-extended {\epsilon}-greedy exploration. *arXiv preprint arXiv:2006.01782*, 2020.

Dabney, W., Barreto, A., Rowland, M., Dadashi, R., Quan, J., Bellemare, M. G., and Silver, D. The value-improvement path: Towards better representations for reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 7160–7168, 2021.

Dai, Y., Luo, H., and Chen, L. Follow-the-perturbed-leader for adversarial markov decision processes with bandit feedback. *Advances in Neural Information Processing Systems*, 35:11437–11449, 2022.

Dann, C., Mansour, Y., Mohri, M., Sekhari, A., and Sridharan, K. Guarantees for epsilon-greedy reinforcement learning with function approximation. In *International conference on machine learning*, pp. 4666–4689. PMLR, 2022.

DiCiccio, T. J. and Efron, B. Bootstrap confidence intervals. *Statistical science*, 11(3):189–228, 1996.

Du, S., Kakade, S., Lee, J., Lovett, S., Mahajan, G., Sun, W., and Wang, R. Bilinear classes: A structural framework for provable generalization in rl. In *International Conference on Machine Learning*, pp. 2826–2836. PMLR, 2021.

Dwaracherla, V., Lu, X., Ibrahimi, M., Osband, I., Wen, Z., and Van Roy, B. Hypermodels for exploration. *arXiv preprint arXiv:2006.07464*, 2020.

Eysenbach, B. and Levine, S. Maximum entropy rl (provably) solves some robust rl problems. *arXiv preprint arXiv:2103.06257*, 2021.

Eysenbach, B., Gupta, A., Ibarz, J., and Levine, S. Diversity is all you need: Learning skills without a reward function. *arXiv preprint arXiv:1802.06070*, 2018.

Fortunato, M., Azar, M. G., Piot, B., Menick, J., Osband, I., Graves, A., Mnih, V., Munos, R., Hassabis, D., Pietquin, O., et al. Noisy networks for exploration. *arXiv preprint arXiv:1706.10295*, 2017.

Foster, D. J., Kakade, S. M., Qian, J., and Rakhlin, A. The statistical complexity of interactive decision making. *arXiv preprint arXiv:2112.13487*, 2021.

Fu, J., Co-Reyes, J. D., and Levine, S. Ex2: Exploration with exemplar models for deep reinforcement learning. *arXiv:1703.01260*, 2017.

Garg, D., Hejna, J., Geist, M., and Ermon, S. Extreme q-learning: Maxent rl without entropy. *arXiv preprint arXiv:2301.02328*, 2023.

Gopalan, A. and Mannor, S. Thompson sampling for learning parameterized markov decision processes. In *Conference on Learning Theory*, pp. 861–898. PMLR, 2015.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

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.

Hong, Z.-W., Shann, T.-Y., Su, S.-Y., Chang, Y.-H., Fu, T.-J., and Lee, C.-Y. Diversity-driven exploration strategy for deep reinforcement learning. *Advances in neural information processing systems*, 31, 2018.

Houthooft, R., Chen, X., Duan, Y., Schulman, J., De Turck, F., and Abbeel, P. Vime: Variational information maximizing exploration. In *NIPS*, 2016.

Huang, S., Dossa, R. F. J., Ye, C., Braga, J., Chakraborty, D., Mehta, K., and Araújo, J. G. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274): 1–18, 2022. URL http://jmlr.org/papers/v23/21-1342.html.

Ishfaq, H., Cui, Q., Nguyen, V., Ayoub, A., Yang, Z., Wang, Z., Precup, D., and Yang, L. Randomized exploration in reinforcement learning with general value function approximation. In *International Conference on Machine Learning*, pp. 4607–4616. PMLR, 2021.

Ishfaq, H., Lan, Q., Xu, P., Mahmood, A. R., Precup, D., Anandkumar, A., and Azizzadenesheli, K. Provable and practical: Efficient exploration in reinforcement learning via langevin monte carlo. *arXiv preprint arXiv:2305.18246*, 2023.

Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., and Kavukcuoglu, K. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016.

Jia, Z., Li, G., Rakhlin, A., Sekhari, A., and Srebro, N. When is agnostic reinforcement learning statistically tractable? *arXiv preprint arXiv:2310.06113*, 2023.

Jin, C., Yang, Z., Wang, Z., and Jordan, M. I. Provably efficient reinforcement learning with linear function approximation. In *Conference on Learning Theory*, pp. 2137–2143. PMLR, 2020.

Jin, C., Liu, Q., and Miryoosefi, S. Bellman eluder dimension: New rich classes of rl problems, and sample-efficient algorithms. *Advances in neural information processing systems*, 34:13406–13418, 2021.

Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1-2):99–134, 1998.

Konda, V. and Tsitsiklis, J. Actor-critic algorithms. *Advances in neural information processing systems*, 12, 1999.

Kveton, B., Szepesvari, C., Ghavamzadeh, M., and Boutilier, C. Perturbed-history exploration in stochastic linear bandits. *arXiv preprint arXiv:1903.09132*, 2019a.

Kveton, B., Szepesvari, C., Vaswani, S., Wen, Z., Lattimore, T., and Ghavamzadeh, M. Garbage in, reward out: Bootstrapping exploration in multi-armed bandits. In *International Conference on Machine Learning*, pp. 3601–3610. PMLR, 2019b.

Kveton, B., Zaheer, M., Szepesvari, C., Li, L., Ghavamzadeh, M., and Boutilier, C. Randomized exploration in generalized linear bandits. In *International Conference on Artificial Intelligence and Statistics*, pp. 2066–2076. PMLR, 2020.

Kveton, B., Konobeev, M., Zaheer, M., Hsu, C.-w., Mladenov, M., Boutilier, C., and Szepesvari, C. Meta-thompson sampling. In *International Conference on Machine Learning*, pp. 5884–5893. PMLR, 2021.

Lattimore, T. and Szepesvári, C. *Bandit algorithms*. Cambridge University Press, 2020.

Li, Z., Li, Y., Zhang, Y., Zhang, T., and Luo, Z.-Q. Hyperdqn: A randomized exploration method for deep reinforcement learning. In *International Conference on Learning Representations*, 2021.

Lyle, C., Rowland, M., Ostrovski, G., and Dabney, W. On the effect of auxiliary tasks on representation dynamics. In *International Conference on Artificial Intelligence and Statistics*, pp. 1–9. PMLR, 2021.

Makoviychuk, V., Wawrzyniak, L., Guo, Y., Lu, M., Storey, K., Macklin, M., Hoeller, D., Rudin, N., Allshire, A., Handa, A., et al. Isaac gym: High performance gpu-based physics simulation for robot learning. *arXiv preprint arXiv:2108.10470*, 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., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. Human-level control through deep reinforcement learning. *Nature*, 2015.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., Silver, D., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In *ICML*, 2016.

Osband, I., Russo, D., and Van Roy, B. (more) efficient reinforcement learning via posterior sampling. *Advances in Neural Information Processing Systems*, 26, 2013.

Osband, I., Blundell, C., Pritzel, A., and Van Roy, B. Deep exploration via bootstrapped dqn. In *NIPS*, 2016a.

Osband, I., Van Roy, B., and Wen, Z. Generalization and exploration via randomized value functions. In *International Conference on Machine Learning*, pp. 2377–2386. PMLR, 2016b.

Ostrovski, G., Bellemare, M. G., Oord, A., and Munos, R. Count-based exploration with neural density models. In *International conference on machine learning*, pp. 2721–2730. PMLR, 2017.

Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the 34th International Conference on Machine Learning*, pp. 2778–2787, 2017.

Pathak, D., Gandhi, D., and Gupta, A. Self-supervised exploration via disagreement. In *International Conference on Machine Learning*, pp. 5062–5071. PMLR, 2019.

Plappert, M., Houthooft, R., Dhariwal, P., Sidor, S., Chen, R. Y., Chen, X., Asfour, T., Abbeel, P., and Andrychowicz, M. Parameter space noise for exploration. *arXiv preprint arXiv:1706.01905*, 2017.

Pritzel, A., Uria, B., Srinivasan, S., Badia, A. P., Vinyals, O., Hassabis, D., Wierstra, D., and Blundell, C. Neural episodic control. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 2827–2836. JMLR. org, 2017.

Rakhlin, A. and Sridharan, K. Bistro: An efficient relaxation-based method for contextual bandits. In *International Conference on Machine Learning*, pp. 1977–1985. PMLR, 2016.

Ramesh, A., Kirsch, L., van Steenkiste, S., and Schmidhuber, J. Exploring through random curiosity with general value functions. *arXiv preprint arXiv:2211.10282*, 2022.

Russo, D. J., Van Roy, B., Kazerouni, A., Osband, I., Wen, Z., et al. A tutorial on thompson sampling. *Foundations and Trends® in Machine Learning*, 11(1):1–96, 2018.

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

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

Schwarke, C., Klemm, V., Van der Boon, M., Bjelonic, M., and Hutter, M. Curiosity-driven learning of joint locomotion and manipulation tasks. In *Proceedings of The 7th Conference on Robot Learning*, volume 229, pp. 2594–2610. PMLR, 2023.

Sun, W., Jiang, N., Krishnamurthy, A., Agarwal, A., and Langford, J. Model-based rl in contextual decision processes: Pac bounds and exponential improvements over model-free approaches. In *Conference on learning theory*, pp. 2898–2933. PMLR, 2019.

Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.

Sutton, R. S., Precup, D., and Singh, S. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2): 181–211, 1999.

Sutton, R. S., Modayil, J., Delp, M., Degris, T., Pilarski, P. M., White, A., and Precup, D. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pp. 761–768, 2011.

Taïga, A. A., Fedus, W., Machado, M. C., Courville, A., and Bellemare, M. G. Benchmarking bonus-based exploration methods on the arcade learning environment. *arXiv preprint arXiv:1908.02388*, 2019.

Thompson, W. R. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3-4):285–294, 1933.

Thompson, W. R. On the theory of apportionment. *American Journal of Mathematics*, 57(2):450–456, 1935.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Vaswani, S., Mehrabian, A., Durand, A., and Kveton, B. Old dog learns new tricks: Randomized ucb for bandit problems. *arXiv preprint arXiv:1910.04928*, 2019.

Williams, R. J. and Peng, J. Function optimization using connectionist reinforcement learning algorithms. *Connection Science*, 3(3):241–268, 1991.

Wu, R. and Sun, W. Making rl with preference-based feedback efficient via randomization. *arXiv preprint arXiv:2310.14554*, 2023.

Zanette, A., Brandfonbrener, D., Brunskill, E., Pirotta, M., and Lazaric, A. Frequentist regret bounds for randomized least-squares value iteration. In *International Conference on Artificial Intelligence and Statistics*, pp. 1954–1964. PMLR, 2020.

Zhang, T. Feel-good thompson sampling for contextual bandits and reinforcement learning. *SIAM Journal on Mathematics of Data Science*, 4(2):834–857, 2022.

Zhang, T., Ren, T., Yang, M., Gonzalez, J., Schuurmans, D., and Dai, B. Making linear mdps practical via contrastive representation learning. In *International Conference on Machine Learning*, pp. 26447–26466. PMLR, 2022.

A. Additional Related Works

There are two main approaches in RL for exploration, (a) Randomize the agent by injecting noise thus inducing diverse behavior, and (b) Provide explicit reward bonuses that incentivize the agent to go to novel states. RLE, bridging the two approaches, injects noise into the agent by adding random latent reward bonuses during training.

Randomness is the key tool in many exploration strategies in RL. Perhaps the most popular example is ϵ-greedy or Boltzmann sampling-based exploration (Mnih et al., 2013; Dann et al., 2022; Cesa-Bianchi et al., 2017; Eysenbach et al., 2018), which explores by playing random actions. Entropy regularization (Williams & Peng, 1991; Mnih et al., 2016), and MaxEnt RL (Haarnoja et al., 2018; Eysenbach & Levine, 2021; Garg et al., 2023; Hazan et al., 2019) are other instances of exploration algorithms that utilize randomness, as they explicitly bias towards learning policies that have a high entropy. Another exploration approach is to directly inject noise into the parameters of the policy or value networks, e.g. in off-policy methods (Fortunato et al., 2017), RLHF with linear MDPs (Wu & Sun, 2023), online RL in tabular (Osband et al., 2016b), and linear MDPs (Zanette et al., 2020). Another line of work includes using Thompson sampling (Thompson, 1933; 1935; Russo et al., 2018), or posterior sampling for exploration (Osband et al., 2013; Gopalan & Mannor, 2015; Kveton et al., 2021; Zhang, 2022), which maintains a posterior distribution over the ground truth model and relies on the uncertainty in the posterior distribution for exploration. Posterior sampling, however, is intractable in practical RL settings due to the need to sample from the extremely complex posterior distribution; Various empirical approaches aim to sample from an approximate posterior instead (Li et al., 2021; Dwaracherla et al., 2020), but are unfortunately memory intensive. We note that RLE is different from these other works as it explores by adding random rewards instead of randomizing over the actions, policies, or models of the environment.

Exploration by adding explicit reward bonuses is also well studied in both theoretical and applied RL literature. A popular technique is to add novelty-based exploration bonuses that are constructed using prediction errors in transition dynamics (Pathak et al., 2017; 2019; Ramesh et al., 2022) or outputs of randomly initialized target network RND (Burda et al., 2019), etc. Other approaches construct reward bonuses using upper confidence bounds on the uncertainty estimates for the underlying model (Auer et al., 2008; Vaswani et al., 2019), using discriminatively trained exemplar models to estimate novelty (Fu et al., 2017), or using elliptical potentials when the MDP has a linear parameterization (Jin et al., 2020; Zhang et al., 2022; Agarwal et al., 2020). Unfortunately, these methods often introduce additional components into the learning setups, e.g. additional neural networks for generating bonuses and the associated hyperparameters, which can make learning unstable. In contrast, RLE is much simpler to deploy as it adds random reward functions that are computed using features from the policy network.

RLE closely resembles the idea of *Follow The Perturbed Leader* (FTPL) developed in RL theory literature (Kveton et al., 2019b;a; 2020; Rakhlin & Sridharan, 2016; Dai et al., 2022). FTPL-based methods explore by adding carefully designed perturbations to the reward function that can guarantee optimism; since the perturbations are closely tied to the underlying modeling assumptions, FTPL-based methods are currently limited to restricted settings like linear bandits, tabular MDPs, etc. In contrast, RLE simply adds a random reward function sampled from a fixed distribution P_z, and is thus applicable in more practical RL settings. Another major difference is that our method also utilizes z-conditioned policies and value functions, and thus the randomness is shared amongst the reward function, policy network and value network.

Finally, there is a long line of work in the theoretical RL literature on developing exploration algorithms that can optimally exploit the structure of the underlying task in model-based RL (Ayoub et al., 2020; Sun et al., 2019; Foster et al., 2021), model-free RL (Jin et al., 2021; Du et al., 2021) and agnostic RL setting (Jia et al., 2023), however, the focus in these works is on statistical efficiency and the provided approaches are not computationally tractable.

B. Experimental Implementation Details

In this section, we provide the hyperparameters and implementation details of our algorithm (RLE) along with the baseline methods (PPO, RND, NOISYNET) for the FOURROOM and ATARI Environments. We also provide hyperparameters and implementation details for all ISAACGYM experiments.

B.1. RLE Pseudocode

Below, we provide the pseudocode for RLE in Algorithm 2.

Algorithm 2 Detailed Pseudocode for Random Latent Exploration (RLE)

Input: Latent distribution $P_{\boldsymbol{z}}$, N parallel workers, T steps per update, S steps per sampling, feature network update rate τ
Randomly initialize a feature network ϕ with the same backbone architecture as the policy and value networks
Initialize running mean $\boldsymbol{\mu} = \mathbf{0}$ and standard deviation $\boldsymbol{\sigma} = \mathbf{1}$ estimates of $\phi(s)$ over the state space
Sample an initial latent vector for each parallel worker: $\boldsymbol{z} \sim P_{\boldsymbol{z}}$
repeat
  Sample initial state s_0.
  for $t = 0, \dots, T$ **do**
    Take action $a_t \sim \pi(.|s_t, \boldsymbol{z})$ and transition to s_{t+1}
    Compute feature $f(s_{t+1}) = (\phi(s_{t+1}) - \boldsymbol{\mu})/\boldsymbol{\sigma}$
    Compute random reward: $F(s_{t+1}, \boldsymbol{z}) = \frac{f(s_{t+1})}{\|f(s_{t+1})\|} \cdot z$
    Receive reward: $r_t = R(s_t, a_t) + F(s_{t+1}, \boldsymbol{z})$
    for $i = 0, 1, \dots, N-1$ **do**
      if worker i terminated **or** S timesteps passed without resampling **then**
        Resample sample $\boldsymbol{z} \sim P_{\boldsymbol{z}}$ for worker i
      end if
    end for
  end for
  Update policy network π and value network V^π with the collected trajectory $(\boldsymbol{z}, s_0, a_0, r_0, s_1, \cdots, s_T)$
  Update feature network ϕ using the value network's parameters: $\phi \leftarrow \tau \cdot \pi + (1-\tau) \cdot \phi$
  Update $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ using the batch of collected experience.
until convergence

B.2. FOURROOM Environment

We provide the hyperparameters used for experiments in the FOURROOM environment in Table 2.

B.2.1. RLE IMPLEMENTATION IN FOURROOM ENVIROMENT

In our implementation of RLE for FOURROOM environment, we ensure that the random reward functions $F(s, z)$ take values in $[-1, 1]$.To compute the reward given a state s and latent variable $\boldsymbol{z}$, we normalize the output of $\phi(s)$ to have unit norm. Specifically, we define the reward as:

$$F(s, \boldsymbol{z}) = \frac{\phi(s)}{\|\phi(s)\|} \cdot z,$$

where ϕ is the randomly initialized feature network that transforms the state s to a vector with the same dimension as $\boldsymbol{z}$. In the FOURROOM environment, we sample $\boldsymbol{z}$ from the unit sphere at every training step, which occurs every 128 timesteps. We perform the sampling independently for each of the 32 parallel workers.

B.3. ATARI

We display the hyperparameters used for experiments in ATARI games in Table 3. For PPO and RND, we use the default hyperparameters based on the `cleanrl` codebase (Huang et al., 2022), which were tuned for ATARI games. For NOISYNET, we use the same hyperparameters as PPO with the exception of the entropy loss weight, which is set to 0 as recommended

by (Fortunato et al., 2017). We give a detailed description of the ATARI implementation of RLE below.

B.3.1. RLE IMPLEMENTATION DETAILS IN ATARI

Feature network architecture and update. We start with a randomly initialized neural network ϕ which takes a state s as input and outputs a vector in $\mathbb{R}^d$, which has the same dimension as $\boldsymbol{z}$. In our implementation, ϕ contains a CNN backbone with an identical architecture to the (shared) policy and value backbone, along with a final linear layer on top to convert it to a low dimension $\mathbb{R}^d$. In our implementation, we choose $d = 8$. To update ϕ, we follow the rule:

$$\text{CNN}_\phi \leftarrow \tau \cdot \text{CNN}_V + (1 - \tau) \cdot \text{CNN}_\phi$$

for a small value of τ, and we choose $\tau = 0.005$ for our experiments. This network update is inspired by the target network update in DQN (Mnih et al., 2015) and does not require any gradient steps.

Computation of random reward bonus. When the agent experiences a transition (s, a, s'), we obtain random reward bonus from ϕ as follows: Obtain the low-dimensional vector output $\phi(s')$. We standardize the output of $\phi(s')$ using a running mean and standard deviation estimate so that the output is a normal distribution on $\mathbb{R}^d$. Meanwhile, sample a vector $\boldsymbol{z} \sim \mathcal{S}^{d-1}$, and compute the following value

$$F(s', \boldsymbol{z}) = \frac{\phi(s') \cdot \boldsymbol{z}}{\|\phi(s')\|}.$$

Policy input. The policy observes the observation returned by the environment, which is 4 stacked grayscale frames of dimension $(84, 84)$. In addition, the policy observes $\boldsymbol{z}$ as well as the random reward $F(s_t, \boldsymbol{z})$ from the previous time step.

Resampling of the latent variable $\boldsymbol{z}$. In our Atari experiments, there are 128 parallel workers. We sample $\boldsymbol{z}$ independently across all workers from the d-dimensional unit sphere $\mathcal{S}^{d-1}$, and resample upon either of the following signals:

1. An environment has reached the 'DONE' flag, or
2. An environment has survived with this $\boldsymbol{z}$ for 1280 time steps.

Policy training. We use PPO with the augmented observation space train on the combined reward as usual. As we resample $\boldsymbol{z}$ during an episode, we also treat the problem of maximizing randomized reward as episodic. Specifically, we set the 'done' signal to True whenever we resample $\boldsymbol{z}$. Thus, we do not use returns from future $\boldsymbol{z}$ within the same episode to estimate the return under the current $\boldsymbol{z}$.

B.3.2. EVALUATION DETAILS IN ATARI

Human normalized score To compute aggregate performance, we first compute the human normalized score for each seed in each environment as $\frac{\text{Agent}_{\text{score}} - \text{Random}_{\text{score}}}{\text{Human}_{\text{score}} - \text{Random}_{\text{score}}}$. After this, we compute the IQM to measure aggregate performance as recommended in (Agarwal et al., 2021) as it is robust to outliers.

Capped human normalized score We use the capped human normalized score (CHNS) (Badia et al., 2020) to measure the aggregate performance of RLE and baselines in Figure 20. To compute the CHNS, we first compute the human normalized score (HNS) of the agent, as done in (Badia et al., 2020), as $\frac{\text{Agent}_{\text{score}} - \text{Random}_{\text{score}}}{\text{Human}_{\text{score}} - \text{Random}_{\text{score}}}$, after which it is clipped to be between 0 and 1. In addition to aggregate metrics, we provide individual mean scores of all methods in all 57 games in Table 6 along with the corresponding learning curves in Figure 19.

Probability of improvement We use the probability of improvement (POI), recommended in (Agarwal et al., 2021), to measure the relative performance between algorithms across all 57 ATARI games.

Bootstrapped confidence intervals We use the bootstrapping method (DiCiccio & Efron, 1996; Agarwal et al., 2021) to estimate the confidence intervals for all aggregated metrics we report, and mean performance for an algorithm in one environment.

B.4. ISAACGYM

We display the hyperparameters used for experiments in IsaacGym in Table 4. For PPO, we use the default hyperparameters recommended by the `cleanrl` codebase (Huang et al., 2022), which were tuned for IsaacGym tasks and can vary across tasks (specifically, using different hyperparameters for the SHADOWHAND and ALLEGROHAND tasks). For RLE, we use the same hyperparameters for each task. We also display the environment-specific hyperparameters in Table 5, which are shared for each training algorithm we consider in our experiments.

B.4.1. RLE IMPLEMENTATION DETAILS IN ISAACGYM

Feature network architecture and update Similar to our implementation of RLE in the ATARI domain, we start with a randomly initialized neural network ϕ that has an MLP backbone with the same architecture as the backbone of the value function. We update the backbone paramaters using the same slow moving average as in Appendix B.3.1 with $\tau = 0.005$:

$$\text{MLP}_\phi \leftarrow \tau \cdot \text{MLP}_V + (1 - \tau) \cdot \text{MLP}_\phi.$$

Computation of random reward bonus We standardize the output of $\phi(s')$ using a running mean and standard deviation estimate so the output approximates a normal distribution on $\mathbb{R}^d$. We sample a vector $\boldsymbol{z} \sim \mathcal{S}^{d-1}$ and compute the reward as:

$$F(s', \boldsymbol{z}) = \phi(s') \cdot \boldsymbol{z}.$$

Note that this is slightly different from the implementation in ATARI, where we divide by $\|\phi(s')\|$. We use reward normalization for RLE in both domains to scale the randomized reward, so both types have a similar effect.

B.4.2. EVALUATION DETAILS IN ISAACGYM

PPO normalized score We use the IQM of the PPO normalized score to compute aggregate performance across 9 different environments in IsaacGym. We compute the PPO normalized score of the agent as $\text{Agent}_{\text{score}}/\text{PPO}_{\text{mean}}$. For example, the mean performance of PPO in a single environment under the PPO normalized score will be 1. We compute the IQM of this metric for 5 seeds across 9 games (or 45 total runs) to aggregate performance.

Other evaluation details Similar to our experiments in the ATARI domain, we use the bootstrapping method to estimate confidence intervals and use the probability of improvement to measure relative performance between different algorithms.

C. Visualizations on FOURROOM

In this section, we provide further results and visualizations for the FOURROOM environment:

RL with task reward. In addition to the reward-free setting, we train all methods in the FOURROOM environment in a sparse-reward setting for 2.5M timesteps. There is a reward of 1 in the bottom-left corner, and the reward is 0 at all other states. We plot the state visitation counts of all methods after 500K and 2.5M timesteps in Figure 12. In addition, we train five seeds in this environment for each method, and find that the average score for RLE and NOISYNET is 0.6, while the average score for RND and PPO is 0. This suggests that the FOURROOM environment is a task that requires exploration as it is difficult for methods that rely on action noise like PPO to achieve any reward.

State visitations.

- Figure 12 shows state visitation counts for all algorithms trained with a sparse task reward which is 1 at the bottom-left state (red '*') and 0 everywhere else.
- Figure 13 shows state visitation counts for all algorithms trained for 500K and 2.5M steps without any task reward.
- Figure 14 shows state visitation counts for RLE trained in a modified version of the environment with stochastic observations within a 2x2 square region of the environment. Through this, we test if RLE is susceptible to the "NoisyTV" problem (Burda et al., 2019).

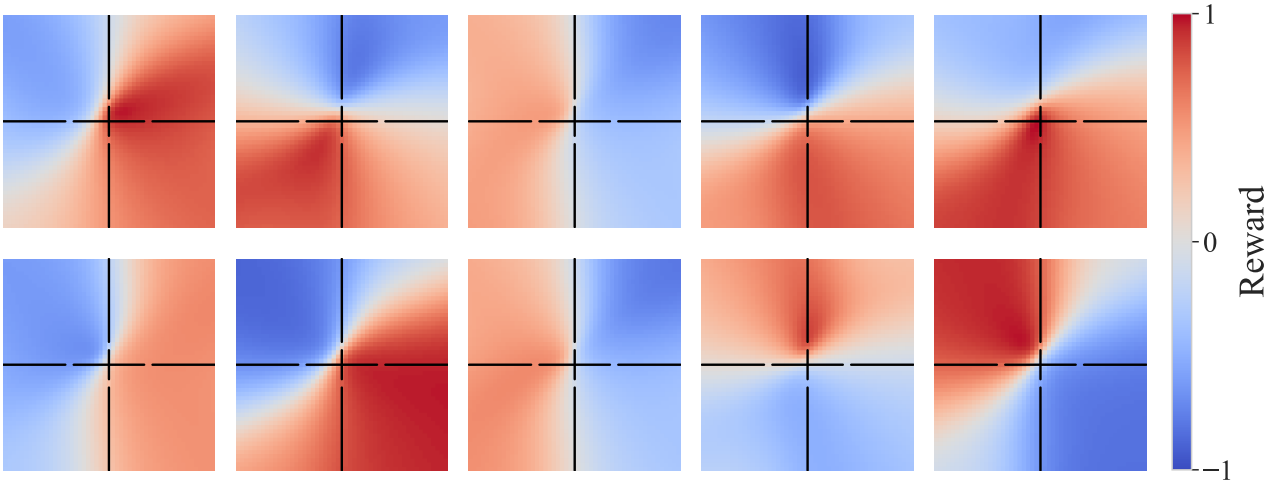
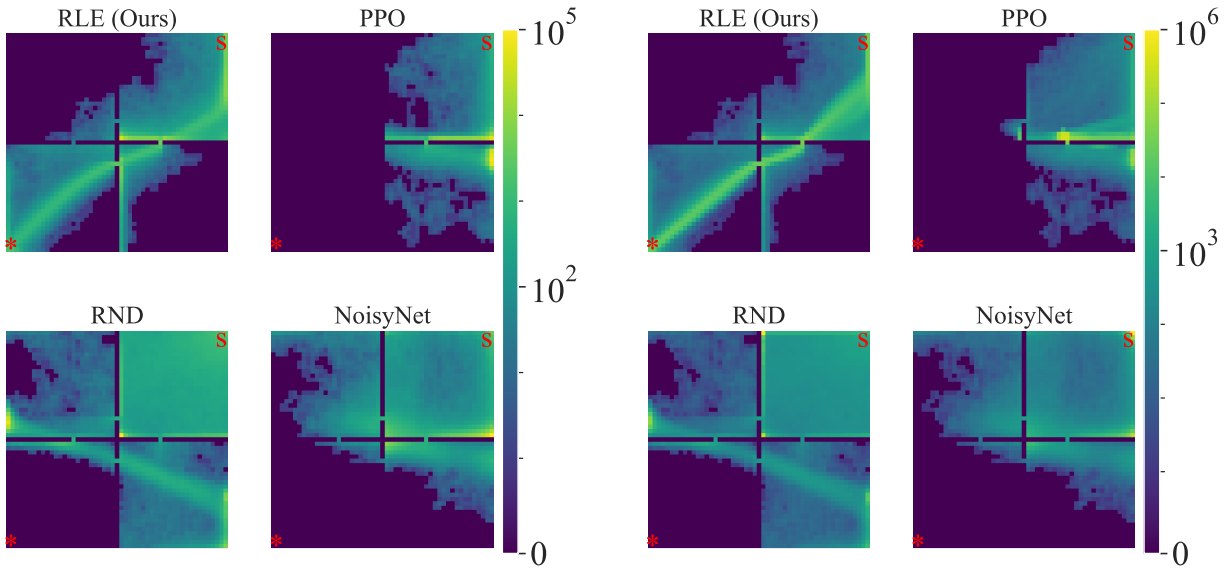


Figure 11. Visualization of the reward function $F(s; \boldsymbol{z})$ for 10 different random choices of $\boldsymbol{z}$ in FOURROOM environment. The reward is is given by $F(s, \boldsymbol{z}) = \boldsymbol{z} \cdot \phi(s)/\|\phi(s)\|$. The above image demonstrates the diversity and coverage of random reward functions in the FOURROOM environment.



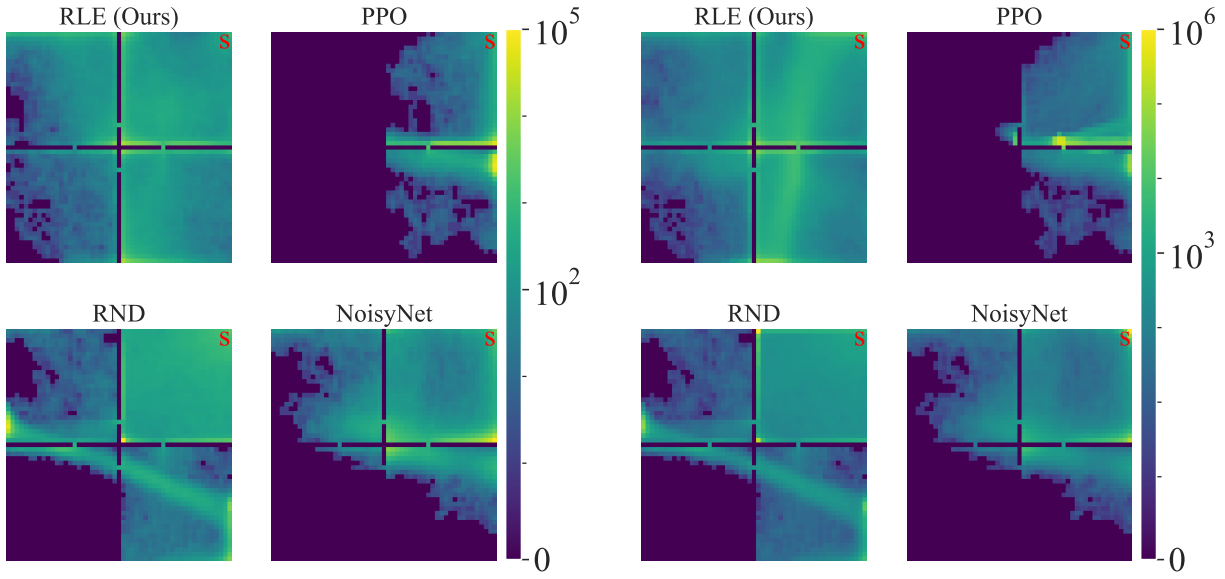
(a) After 500K timesteps.

(b) After 2.5M timesteps.

Figure 12. State visitation counts for different methods on FOURROOMS environment trained for 500K and 2.5M timesteps with task reward. The start location is the top-right state of the grid (represented by the red 'S'). The agent gets a task reward of 1 at the bottom-left state (represented by red '*').

Visualizations of trajectory diversity across algorithms.

- Figure 15 shows 5 trajectories sampled from policies trained with RLE across 5 different seeds at three different points in training: after 500K steps, 1.5M steps, and 2.5M steps.
- Figure 16 shows the same for NOISYNET.



(a) After 500K steps

(b) After 2.5M steps

Figure 13. State visitation counts for different algorithms on FOURROOMS environment after training for 500K timesteps and 2.5M timesteps. All algorithms were trained without task reward (reward-free exploration).

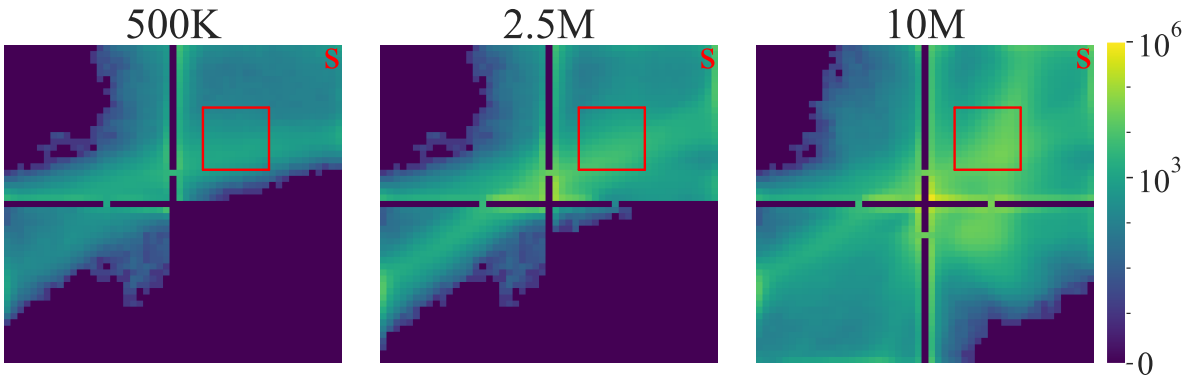


Figure 14. State visitation counts for RLE when trained in an environment with stochasticity in the observation space. The observation is only stochastic within the red square and is deterministic everywhere else. Even after discovering the red square, the agent is able to discover states outside of those regions and continues to explore throughout training. This suggests that RLE is less affected by the NoisyTV problem compared to novelty-based exploration methods.

- Figure 17 shows the same for RND.
- Figure 18 shows the same for PPO.

From visual evaluation, the above plots suggest that RLE induces more diverse trajectories as compared to other baselines (PPO, RND, and NOISYNET) on the FOURROOM environment.

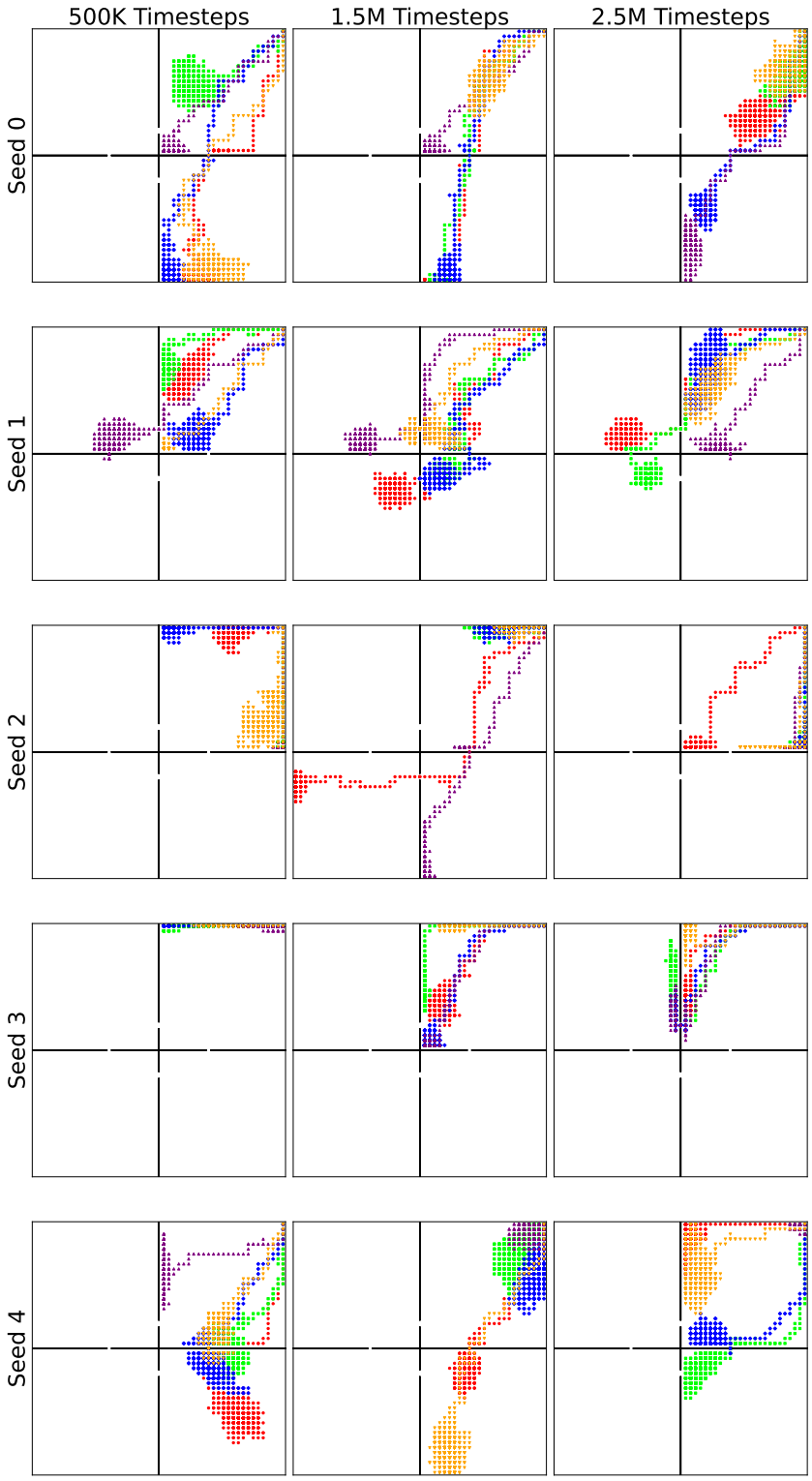


Figure 15. Visualization of trajectories generated by sampling from a policy trained with RLE for 2.5M timesteps in a reward-free setting across 5 seeds at different points in training. We sample 5 trajectories for each seed.

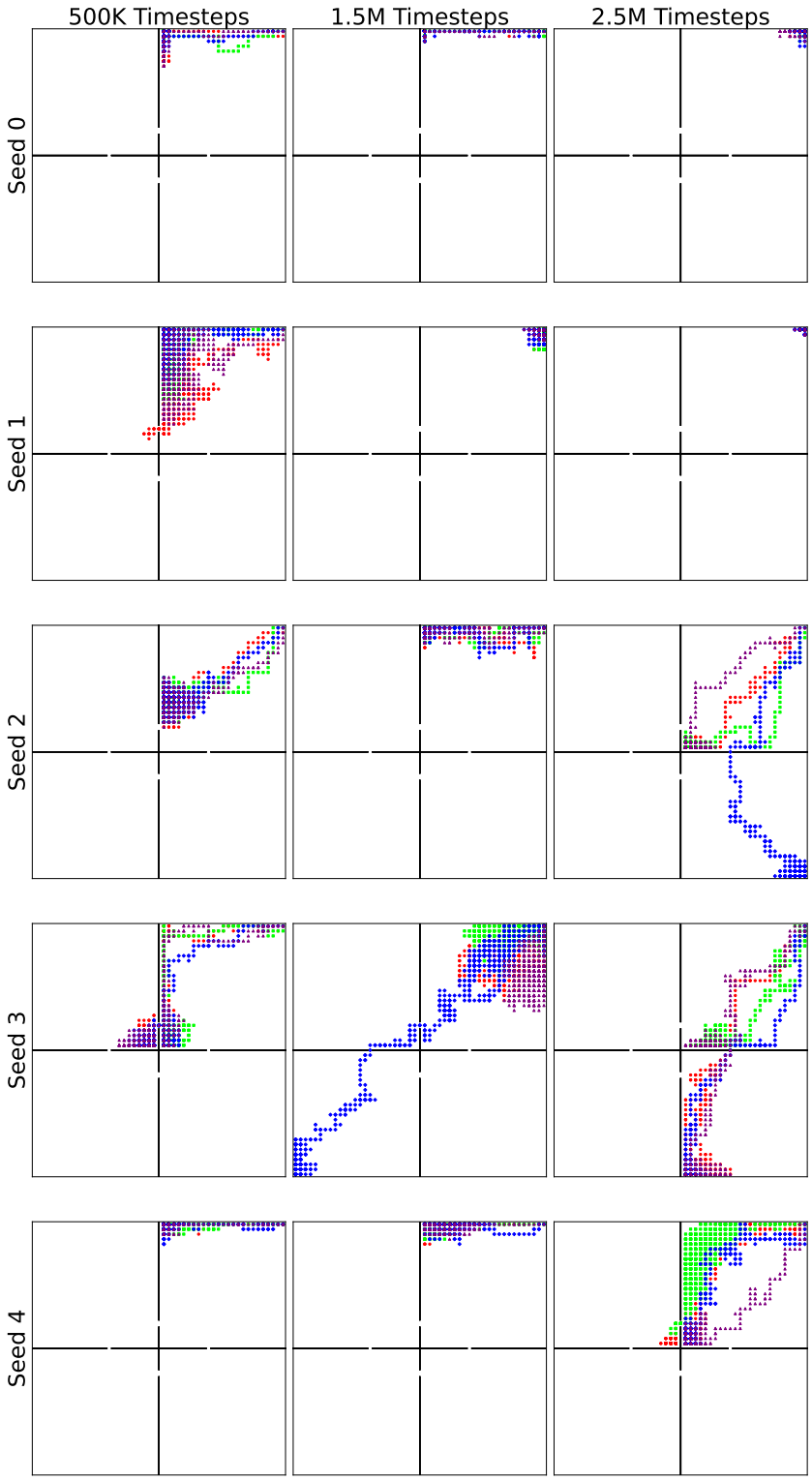


Figure 16. Visualization of trajectories generated by sampling from a policy trained with NOISYNET for 2.5M timesteps in a reward-free setting across 5 seeds at different points in training. We sample 5 trajectories for each seed.

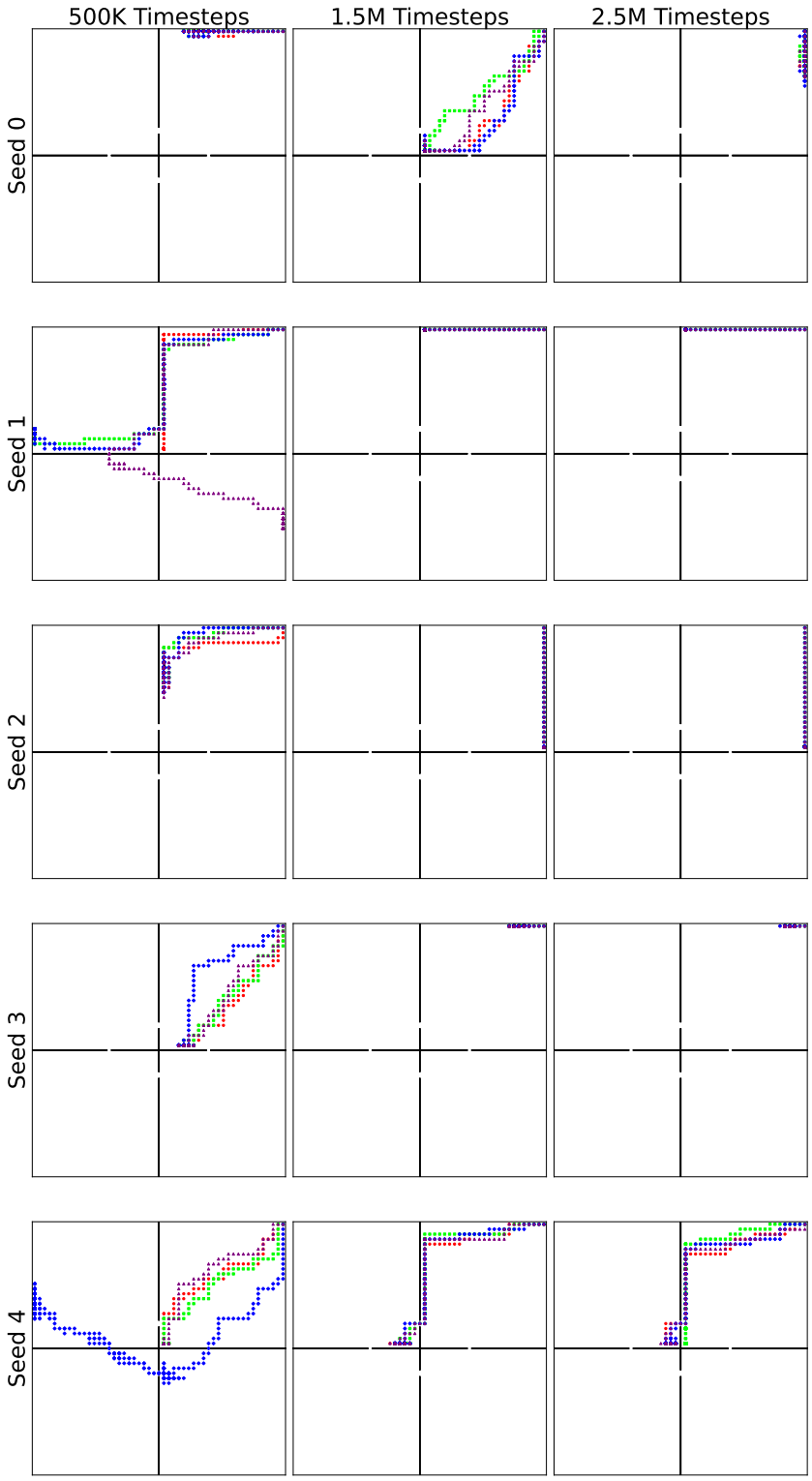


Figure 17. Visualization of trajectories generated by sampling from a policy trained with RND for 2.5M timesteps in a reward-free setting across 5 seeds at different points in training. We sample 5 trajectories for each seed.

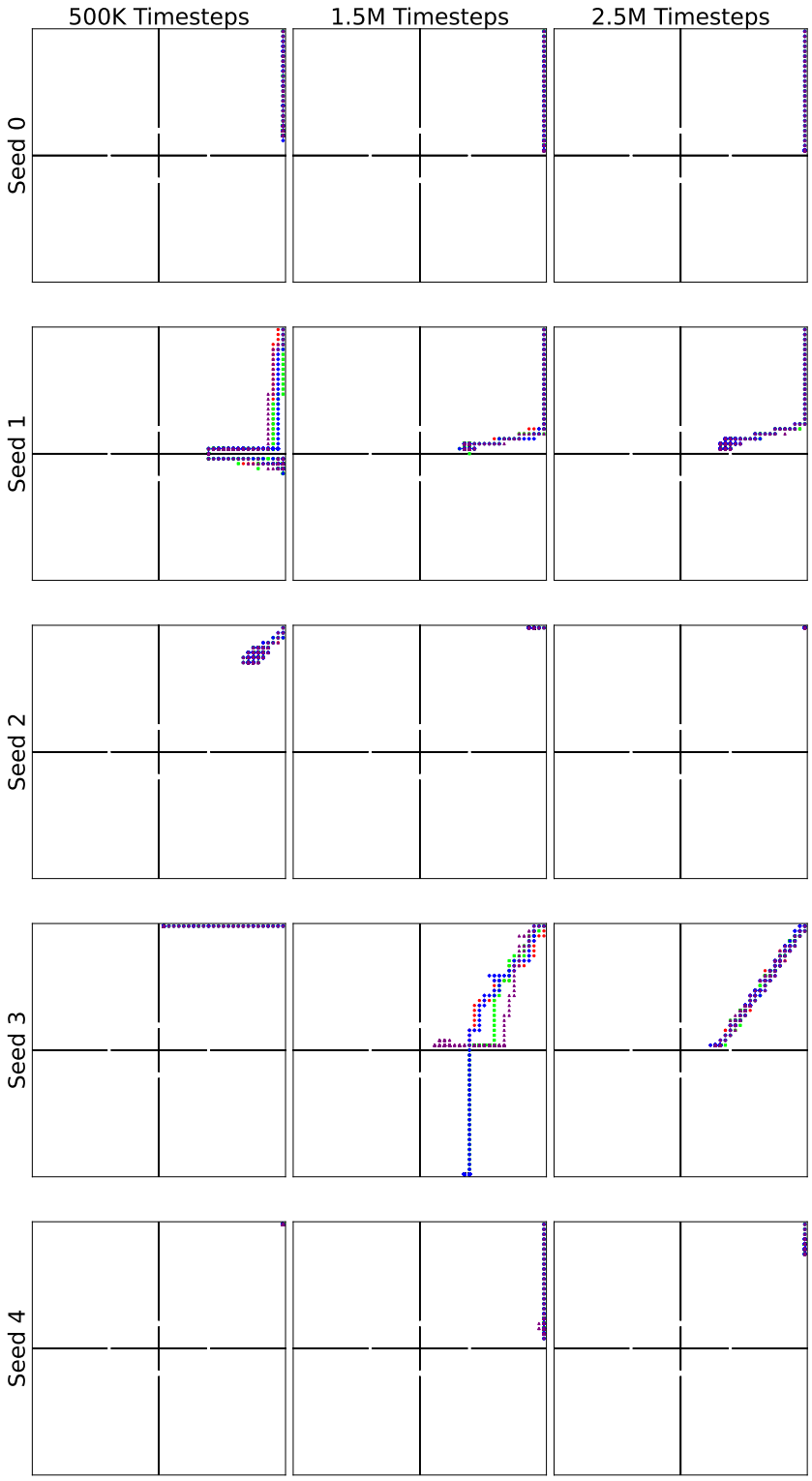


Figure 18. Visualization of trajectories generated by sampling from a policy trained with PPO for 2.5M timesteps in a reward-free setting across 5 seeds at different points in training. We sample 5 trajectories for each seed.

Parameter	Value
PPO	
Total Timesteps	$2,500,000$
Optimizer	Adam
Learning Rate	0.001
Adam Epsilon	0.00001
Parallel Workers	32
Steps per Batch	128
Discount Rate	0.99
Generalized Advantage Estimation λ	0.95
Minibatches per Epoch	4
Epochs per Training Step	4
Clipping Coefficient	0.2
Entropy Loss Weight	0.01
Discount Rate	0.99
Value Loss Weight	0.5
Gradient Norm Bound	0.5
Use Advantage Normalization	True
Use Clipped Value Loss	True
Policy Network Architecture	MLP (64,64,4)
Value Network Architectures	MLP (64,64,1)
Network Activation	Tanh
NOISYNET	
Initial σ	0.017
RND	
Intrinsic Reward Coefficient	1.0
Drop Probability	0.25
Predictor Network Architecture	MLP (256, 256, 256, 256, 256)
Target Network Architecture	MLP (64,256)
Network Activation	ReLU
RLE	
Intrinsic Reward Coefficient	0.1
Latent Vector Dimension	4
Feature Network Architecture	MLP (64,64,64,4)
Network Activation	ReLU

Table 2. Hyperparameters and network architectures for FOURROOM experiments.

Parameter	Value
PPO	
Total Timesteps	$40,000,000$
Optimizer	Adam
Learning Rate	0.0001
Adam Epsilon	0.00001
Parallel Workers	128
Steps per Batch	128
Discount Rate	0.99
Generalized Advantage Estimation λ	0.95
Minibatches per Epoch	4
Epochs per Training Step	4
Clipping Coefficient	0.1
Entropy Loss Weight	0.01
Discount Rate	0.99
Value Loss Weight	0.5
Gradient Norm Bound	0.5
Use Advantage Normalization	True
Use Clipped Value Loss	True
Policy Network Architecture	CNN + MLP (256,448,448,18)
Value Network Architectures	CNN + MLP (256,448,448,1)
Network Activation	ReLU
NOISYNET	
Initial σ	0.017
Entropy Loss Weight	0
RND	
Intrinsic Reward Coefficient	1.0
Extrinsic Reward Coefficient	2.0
Drop Probability	0.25
Discount Rate	0.999
Entropy Loss Weight	0.001
Intrinsic Discount Rate	0.99
Predictor Network Architecture	CNN + MLP (512,512,512)
Target Network Architecture	CNN + MLP (512)
Network Activation	LeakyReLU
RLE	
Intrinsic Reward Coefficient	0.01
Latent Vector Dimension	8
Latent Vector Resample Frequency	1280
Learning Rate	0.0003
Feature Network Update Rate τ	0.005
Feature Network Architecture	CNN + MLP (256,448, 8)
Network Activation	ReLU

Table 3. Hyperparameters and network architectures for ATARI experiments.

Parameter	Value
PPO	
Optimizer	Adam
Learning Rate	0.0026
Adam Epsilon	0.00001
Steps per Batch	16
Discount Rate	0.99
Generalized Advantage Estimation λ	0.95
Minibatches per Epoch	2
Epochs per Training Step	4
Clipping Coefficient	0.2
Entropy Loss Weight	0.0
Discount Rate	0.99
Value Loss Weight	2.0
Gradient Norm Bound	1.0
Use Advantage Normalization	True
Use Clipped Value Loss	False
Policy Network Architecture	MLP (256,256,256)
Value Network Architecture	MLP (256,256,256,1)
Network Activation	Tanh
Reward Scale	1.0
RLE	
Intrinsic Value Loss Weight	0.5
Intrinsic Reward Coefficient	0.01
Latent Vector Dimension	32
Latent Vector Resample Frequency	16
Learning Rate	0.0001
Feature Network Update Rate τ	0.005
Policy Network Architecture	MLP (256,256,256)
Value Network Architecture	MLP (512,512,256,1)
Feature Network Architecture	MLP (512,512,256)
Network Activation	Tanh
PPO (ALLEGROHAND and SHADOWHAND)	
Steps per Batch	8
Minibatches per Epoch	4
Epochs per Training Step	5
Reward Scale	0.01

Table 4. Hyperparameters and network architectures for IsaacGym experiments. The number of training steps and parallel workers depends on the environment, but are shared across different methods.

Parameter	Value
ALLEGROHAND	
Number of Timesteps	$600,000,000$
Number of Parallel Environments	$8,192$
SHADOWHAND	
Number of Timesteps	$600,000,000$
Number of Parallel Environments	$8,192$
BALLBALANCE	
Number of Timesteps	$200,000,000$
Number of Parallel Environments	$4,096$
HUMANOID	
Number of Timesteps	$200,000,000$
Number of Parallel Environments	$4,096$
ANT	
Number of Timesteps	$100,000,000$
Number of Parallel Environments	$4,096$
CARTPOLE	
Number of Timesteps	$100,000,000$
Number of Parallel Environments	$4,096$
FRANKACABINET	
Number of Timesteps	$100,000,000$
Number of Parallel Environments	$4,096$
ANYMAL	
Number of Timesteps	$100,000,000$
Number of Parallel Environments	$4,096$
ANYMALTERRAIN	
Number of Timesteps	$100,000,000$
Number of Parallel Environments	$4,096$

Table 5. Environment-specific parameters and their values. These parameters are shared across all algorithms.

D. Detailed Results and Learning Curves on all ATARI Games

We provide:

- The scores for each of the algorithms (RLE (Ours), PPO, RND and NOISYNET) on all 57 ATARI games in Table 6. Each of the algorithms was trained for 40M steps on all Atari games except for the results for MONTEZUMA'S REVENGE where we trained for 400M steps. The reported Human performance is obtained from Mnih et al. (2013); Badia et al. (2020).
- Learning curves for all the algorithms (RLE (Ours), PPO, RND and NOISYNET) for all 57 ATARI games in Figure 19.
- Aggregated capped human normalized score (described in Appendix B.3.2) for each of the algorithms (RLE (Ours), PPO, RND and NOISYNET) over all 57 Atari games.
- An ablation study of how the soft update rule for the feature network affects performance on ATARI games taking three metrics into account (scores on individual games, IQM of human normalized score, and probability of improvement over PPO).

E. Detailed Results and Learning Curves on all ISAACGYM Tasks

We provide:

- The learning curves for PPO and RLE in all 9 ISAACGYM tasks that we consider in Figure 23.
- An ablation study of how different network architectures for ϕ affect performance on ISAACGYM tasks IQM of PPO normalized score, and probability of improvement over PPO). We plot the IQM of PPO normalized score in Figure 24, and probability of improvement over PPO in Figure 25.
- An ablation study of how using white noise for randomizing rewards affects performance, shown in Figure 26.

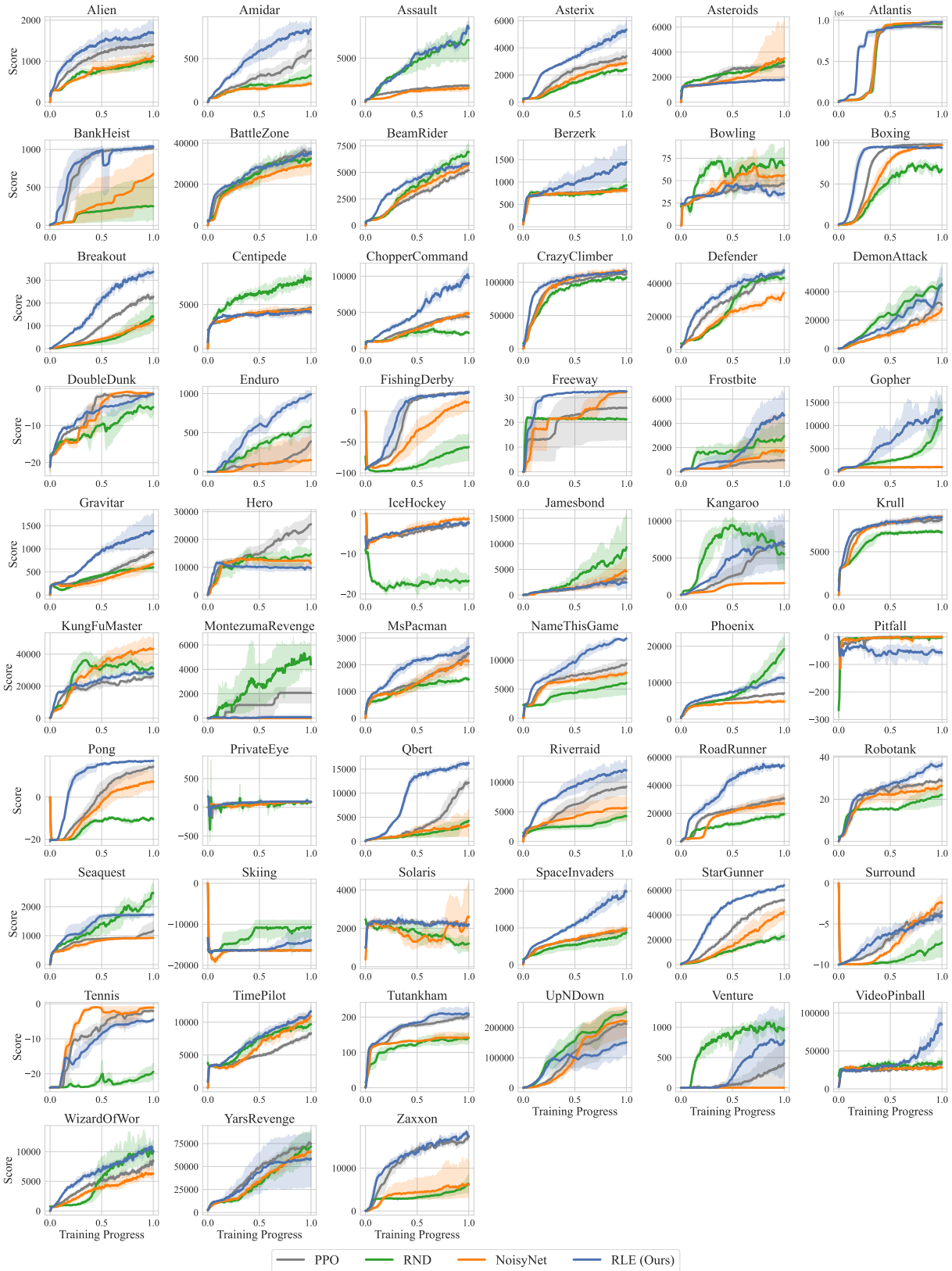


Figure 19. Learning curves for different algorithms for all 57 ATARI games.

	PPO	RND	NoisyNet	RLE (Ours)
Alien-v5	1409.22	1010.94	1112.59	**1680.13**
Amidar-v5	595.53	304.01	210.88	**836.01**
Assault-v5	1886.68	7045.97	1605.63	**8368.57**
Asterix-v5	3392.84	2439.56	2892.50	**5350.79**
Asteroids-v5	2913.15	3232.00	**3513.02**	1798.64
Atlantis-v5	915703.86	957537.41	972286.70	**979023.58**
BankHeist-v5	1017.55	250.91	678.44	**1036.61**
BattleZone-v5	**35832.79**	32536.65	30021.64	34793.65
BeamRider-v5	5200.10	**6915.42**	5755.02	5895.93
Berzerk-v5	845.95	931.95	812.45	**1445.49**
Bowling-v5	46.84	**67.17**	56.17	35.88
Boxing-v5	**97.99**	67.44	97.06	93.94
Breakout-v5	227.34	139.41	127.66	**337.07**
Centipede-v5	4634.04	**7972.15**	4419.47	4151.10
ChopperCommand-v5	4342.80	2140.08	4787.63	**9710.00**
CrazyClimber-v5	112278.70	107602.05	**116637.17**	115593.01
Defender-v5	46957.65	43387.18	34448.07	**47872.91**
DemonAttack-v5	30714.39	44342.27	28287.09	**45217.82**
DoubleDunk-v5	-1.57	-5.04	**-1.43**	-1.51
Enduro-v5	387.89	595.65	150.38	**990.95**
FishingDerby-v5	**31.13**	-57.88	14.14	30.68
Freeway-v5	25.83	21.19	32.40	**32.49**
Frostbite-v5	949.08	2944.37	1747.85	**4658.90**
Gopher-v5	1020.40	11822.26	1055.82	**13290.12**
Gravitar-v5	920.19	597.42	674.99	**1381.69**
Hero-v5	**25495.80**	14695.30	11433.06	9668.68
IceHockey-v5	-2.09	-16.70	**-1.34**	-2.39
Jamesbond-v5	3157.81	**9347.30**	4633.37	2452.21
Kangaroo-v5	6504.67	5474.45	1596.99	**6992.13**
Krull-v5	8731.23	7264.60	**9063.52**	8981.43
KungFuMaster-v5	26131.84	30902.44	**43341.34**	27813.32
MontezumaRevenge-v5	2077.03	**4406.79**	0.00	79.48
MsPacman-v5	2417.82	1446.32	2127.62	**2676.20**
NameThisGame-v5	9392.45	6078.34	7818.62	**13701.36**
Phoenix-v5	7137.14	**19195.54**	4786.92	11272.80
Pitfall-v5	-0.67	-3.41	**-0.05**	-57.65
Pong-v5	14.52	-10.34	7.10	**17.17**
PrivateEye-v5	**98.34**	87.24	95.55	97.79
Qbert-v5	12168.36	4300.73	3381.40	**16261.59**
Riverraid-v5	9268.85	4267.51	5642.73	**12009.63**
RoadRunner-v5	30354.36	19452.68	27037.68	**53920.12**
Robotank-v5	28.68	22.11	26.34	**36.71**
Seaquest-v5	1172.22	**2463.42**	920.71	1724.96
Skiing-v5	-16370.14	**-10644.07**	-16398.72	-13887.77
Solaris-v5	2203.41	1206.94	**2584.66**	2203.76
SpaceInvaders-v5	938.00	878.91	981.24	**1981.37**
StarGunner-v5	52219.39	23174.16	42645.43	**64011.13**
Surround-v5	-3.41	-7.28	**-2.41**	-3.91
Tennis-v5	-2.03	-19.44	**-1.06**	-4.49
TimePilot-v5	8319.51	9695.60	10888.94	**11636.24**
Tutankham-v5	204.57	140.97	142.70	**209.23**
UpNDown-v5	212171.29	**251442.66**	219951.36	151036.48
Venture-v5	401.20	**969.00**	0.06	782.98
VideoPinball-v5	32654.24	35275.58	28236.75	**84825.64**
WizardOfWor-v5	8355.05	**10151.69**	6306.86	9942.29
YarsRevenge-v5	**74833.17**	71789.37	65902.61	58507.98
Zaxxon-v5	17354.21	6273.86	6104.86	**17403.15**

Table 6. Performance on all 57 ATARI games. Each algorithm was trained for 40M timesteps, except for MONTEZUMA'S REVENGE where we trained for 400M timesteps. The reported Human performance is obtained from Mnih et al. (2013); Badia et al. (2020).

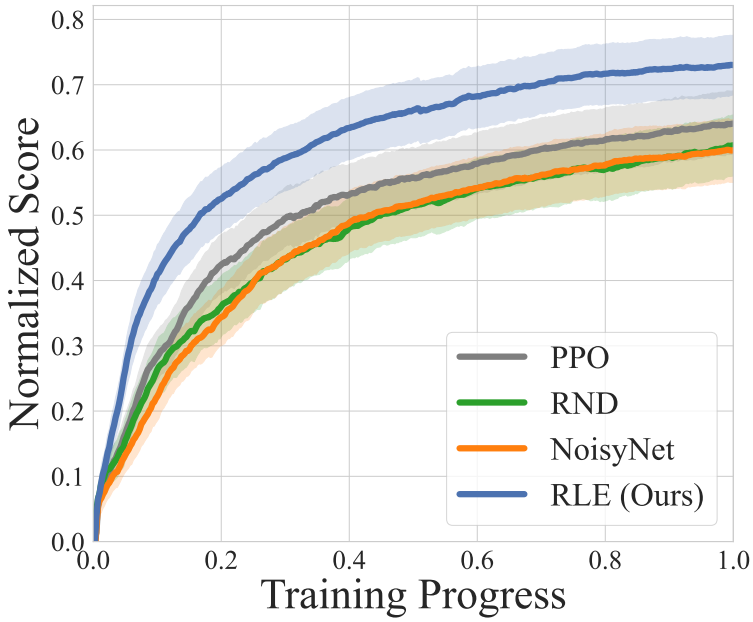
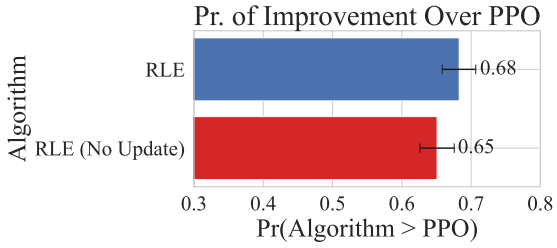
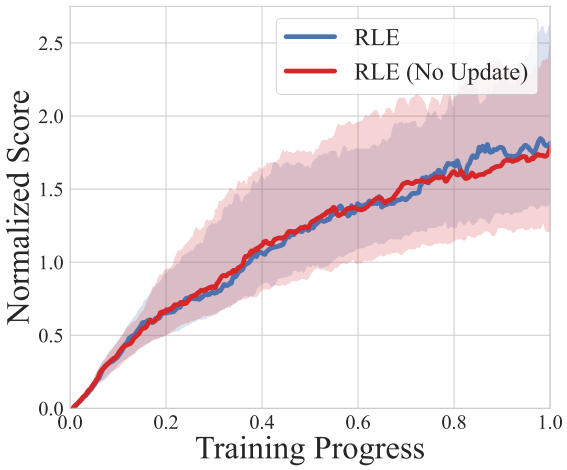


Figure 20. Capped human normalized score across all 57 Atari games. RLE outperforms all other methods in this metric and requires half the training time to reach the same score as the next best method (PPO).



(a) Probability of improvement over PPO with and without a slow value feature update rule. Using the value features leads to a slight increase in performance.



(b) IQM of human normalized score of RLE, both with and without a slow value feature update rule. With respect to this metric, both versions of the method perform very similarly overall.

Figure 21. Comparison of RLE performance with and without a slow value feature update rule.

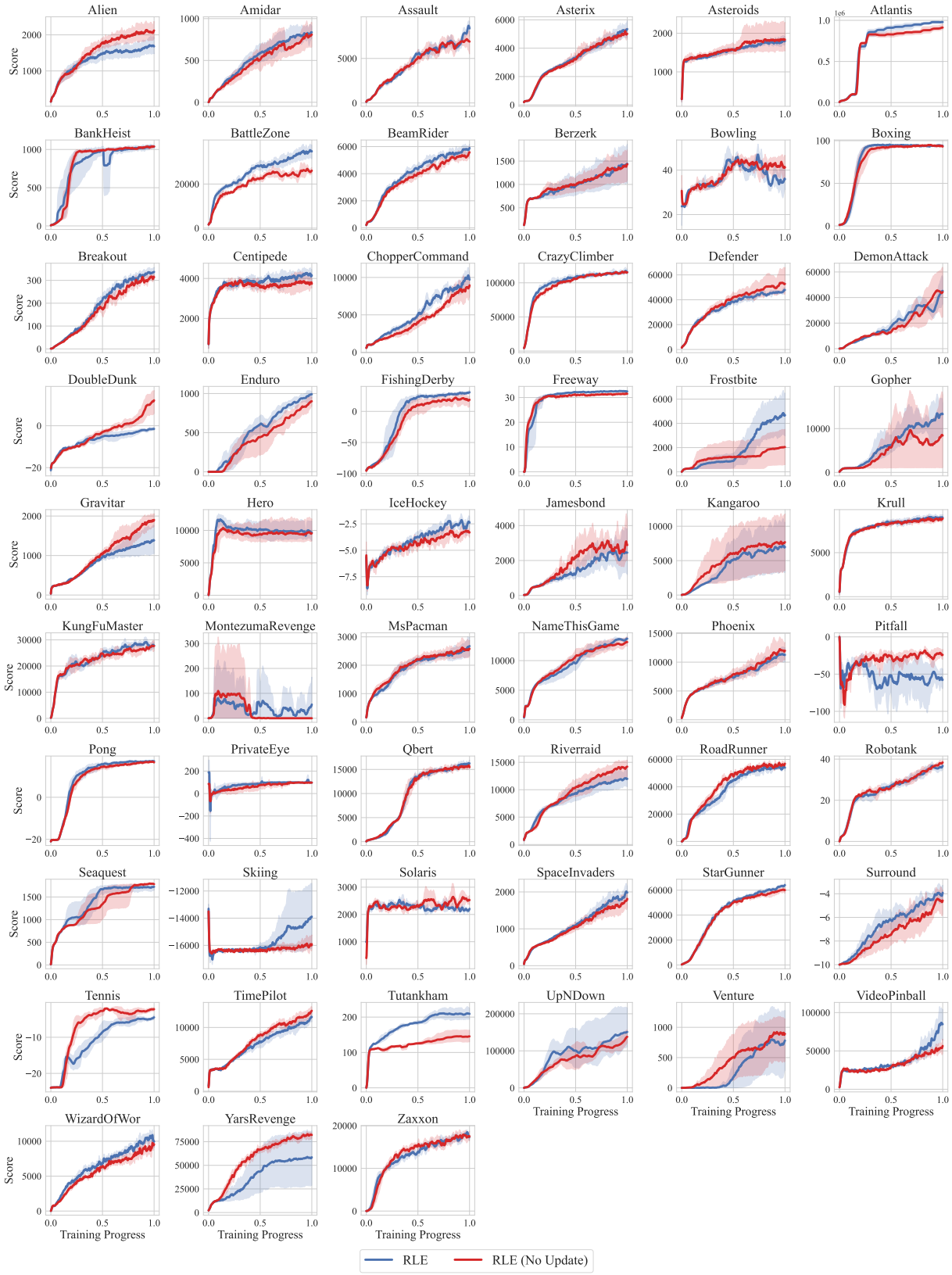


Figure 22. Learning curves for our method with and without a slow value feature update rule. Performance is usually similar.

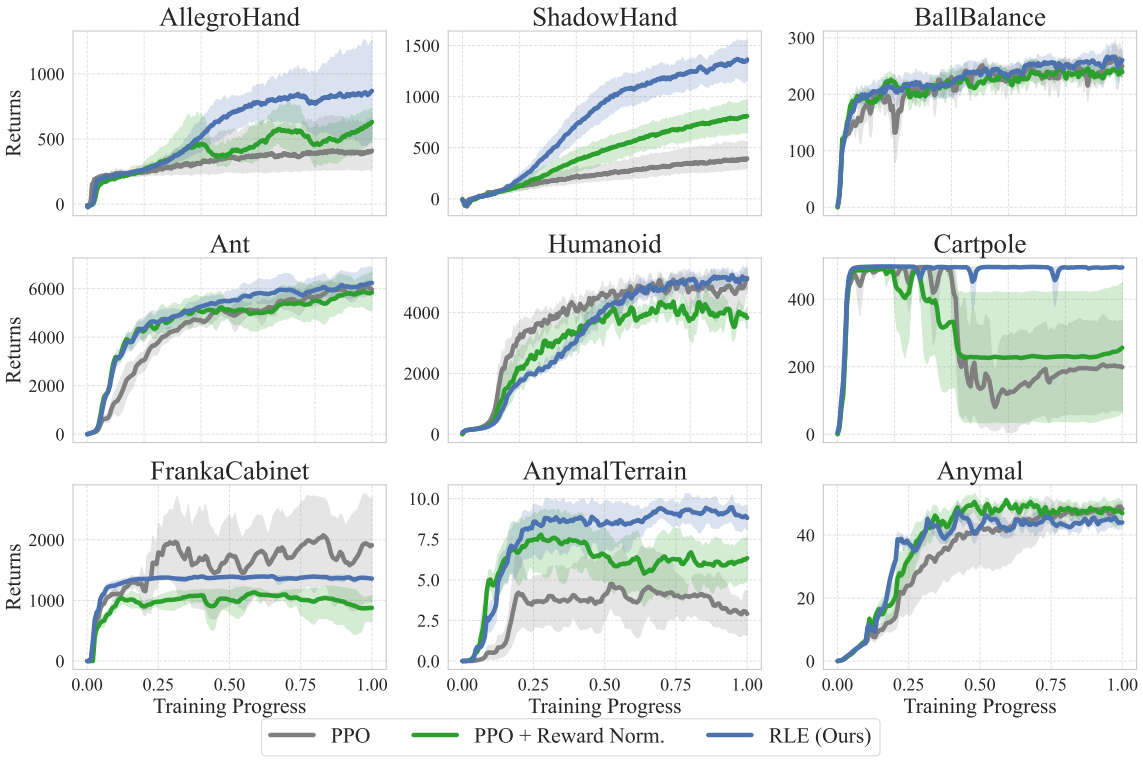


Figure 23. Comparison of learning curves between RLE and standard PPO. RLE achieves return greater than or equal to that of standard PPO in the majority of tasks. We also compare RLE to an ablation of PPO that uses reward normalization and find that RLE improves over it as well.

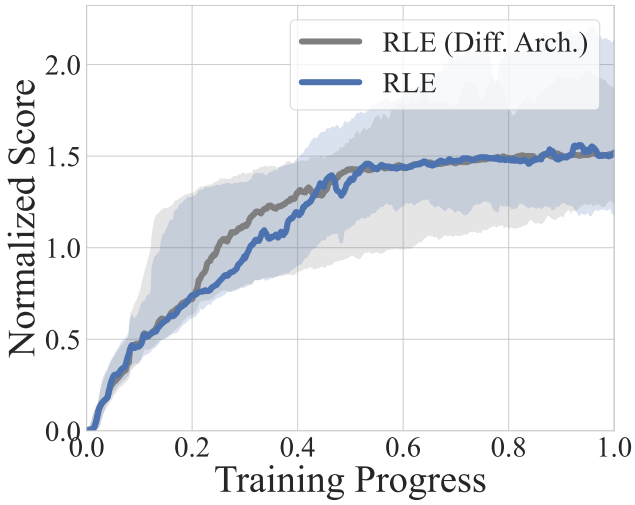


Figure 24. IQM of PPO normalized score of RLE and RLE with a different architecture for ϕ. The different architecture used in this experiment has less width and uses one less layer. The IQM of normalized score is similar for both methods, suggesting that RLE does not highly depend on the architecture of the network ϕ.

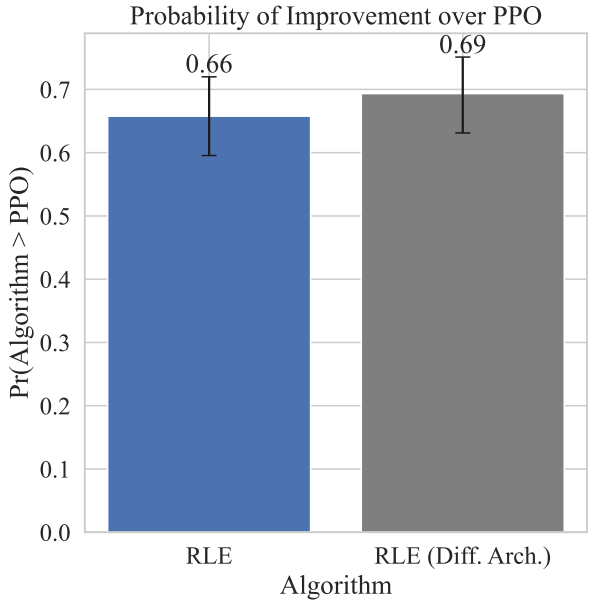


Figure 25. Probability of improvement over PPO of RLE and RLE with a different architecture for ϕ. The probability of improvement for both RLE variants is close and the confidence intervals for the probability of improvement metric heavily overlap. This suggests that RLE is robust to the choice of architecture for ϕ.

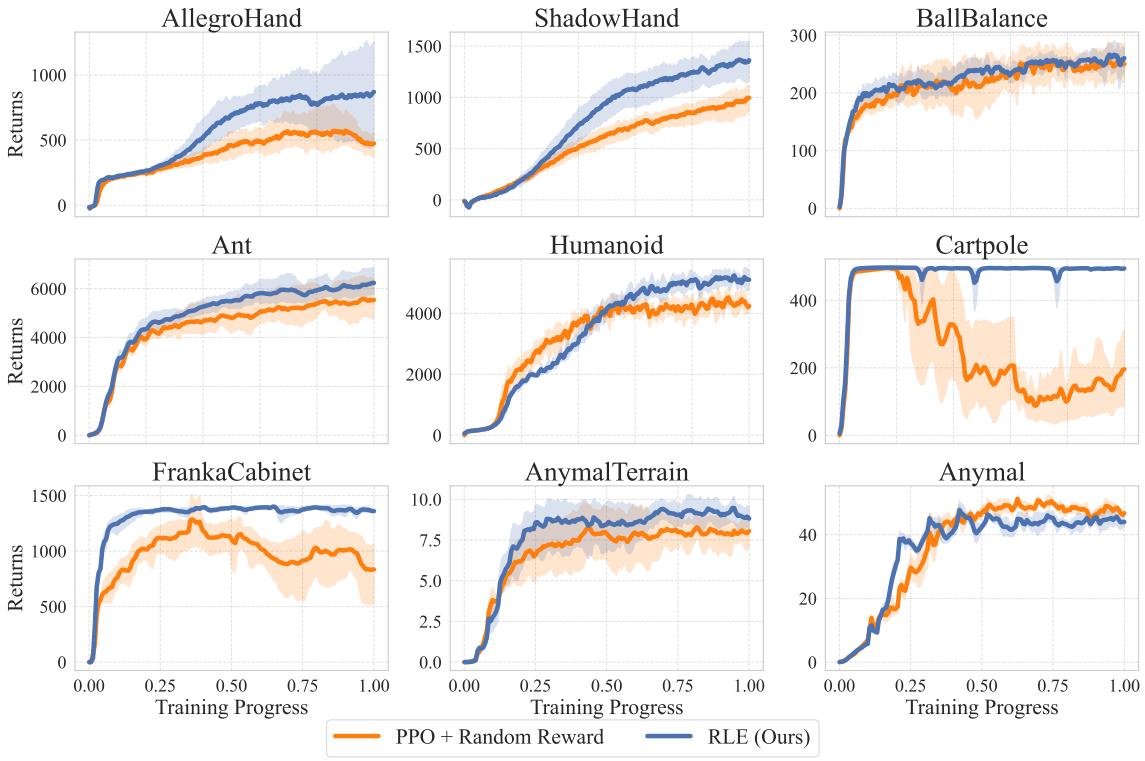


Figure 26. Comparison of learning curves between RLE and PPO with random normal noise sampled i.i.d from a standard normal distribution added to the reward at each timestep. The intrinsic reward coefficient is 0.01. RLE outperforms this variant of PPO in a large majority of games, suggesting that RLE benefits from using state-dependent random rewards.