STM JOURNALS

Current Trends in
Signal Processing

https://journals.stmjournals.com/ctsp

ISSN: 2277-6176 (Online)
ISSN: 2321-4252 (Print)
Volume 15, Issue 3, 2025
September—December
DOI (Journal): 10.37591/CTSP

Research CTSP

Randomized Latent Vectors for Enhanced Reinforcement Learning Exploration

Nagajayant Nagamani*

Abstract

This Study explores random latent exploration (RLE), a cutting-edge reinforcement learning technique designed to significantly enhance exploration by conditioning both policy and value networks on randomly sampled latent vectors. RLE works by sampling a latent vector from a chosen distribution at the start of each trajectory and augmenting the traditional extrinsic reward with an intrinsic reward, computed as the dot-product between state-feature representations and the latent vector. The policy is aware of the latent vector—akin to universal value function conditioning—and optimizes the combined extrinsic plus intrinsic reward signal. To thoroughly evaluate RLE, we benchmark its performance on diverse environments: the discrete-control FourRoom, continuous control tasks in the Isaac Gym suite, and visually complex Atari games. RLE is applied on top of standard proximal policy optimization (PPO), retaining general hyperparameters across all tasks, and compared against well-known exploration baselines such as PPO without exploration bonuses, NoisyNet, and random network distillation (RND). Our results demonstrate that RLE delivers consistently stronger exploration capabilities and higher cumulative reward, particularly in sparse-reward scenarios. We observe broader state-space coverage, quantified by Shannon entropy of visitation distributions, showing that RLE drives deeper and more varied trajectories than the baselines. Furthermore, we analyze the impact of the latent vector distribution. Empirical results indicate that standard normal and von Mises latent sampling outperform uniform and exponential distributions. Building on this insight, we develop two dynamic, adaptive variants—a queue-based scheme and a neural-adaptive von Mises-Fisher variant—which allow the latent distribution to evolve during training and yield even stronger results in complex tasks. Overall, this work establishes RLE as a potent yet easy-to-implement plug-in strategy that combines the simplicity of noise-based methods with the rich exploration benefits of intrinsic reward bonuses, while also stimulating avenues for further research in latent-guided exploration for reinforcement learning.

Keywords: Random network distillation (RND), reinforcement learning, NoisyNet, random latent exploration (RLE), Isaac Lab

INTRODUCTION

Random latent exploration (RLE) is an algorithm that aims to efficiently explore high-dimensional state spaces. Mahankali et al. [1] showed that RLE works for both discrete and continuous action spaces. They compared RLE with standard proximal policy optimization [2], NoisyNet [3], and random network distillation [4].

Exploration is one of the major challenges in reinforcement learning (RL), and it can generally be divided into two categories: noise-based exploration and bonus-based exploration. There are advantages and disadvantages for both, but we have to keep in mind that always choosing the highest short-term

***Author for Correspondence**
Nagajayant Nagamani
E-mail: nagajayant@live.com

Manager, Software Engagement, Virtusa Limited, Chennai, Tamil Nadu, India

Received Date: July 05, 2025
Accepted Date: July 25, 2025
Published Date: August 08, 2025

Citation: Nagajayant Nagamani. Randomized Latent Vectors for Enhanced Reinforcement Learning Exploration. Current Trends in Signal Processing. 2025; 15(3): 14–25p.

(local) reward does not necessarily yield the highest long-term (global) reward. For example, picture the environment. Here, the agent starting in the blue state will always choose the small reward of 1, i.e., going right, then move back to initial state, then move right, and so on, dithering between these two states, instead of accepting to collect a reward of 0 by going left to be able to collect the much higher reward of 100 in the following step.

Mahankali et al. [1] used RND to represent bonus-based exploration, NoisyNet to represent noise-based exploration, and the standard PPO as the baseline benchmark.

SCOPE OF REPRODUCIBILITY

The main claims made by Mahankali et al. [1] that we will test are as follows:

1. As stated in Mahankali et al. [1], RLE consistently achieves a higher overall performance across a range of tasks when compared to baseline methods such as PPO, NoisyNet, and RND. These evaluations span multiple environments, including Atari, Isaac Gym, and FourRoom, and cover both discrete and continuous control tasks.
2. The authors observed that introducing stochasticity into the reward signals used to condition both the policy and value networks encourages the agent to exhibit more diverse behaviors. This, in turn, promotes broader state space exploration owing to the incentive structure formed by random rewards [1].
3. In the discrete-state, discrete action FourRoom environment (with no predefined goal), the study reports that PPO tends to concentrate its state visitation on the initial region (top-right room). In contrast, methods such as RLE, NoisyNet, and RND demonstrated significantly better exploratory behavior, traversing all four rooms, thereby validating their capacity for deep exploration.
4. The performance benefits of RLE over PPO remain consistent, regardless of the distribution from which the *d*-dimensional latent random vector is sampled.
5. The intrinsic reward function must maintain a meaningful correlation with the visited states. If the reward signal appears as unstructured white noise, it can adversely impact the effectiveness of RLE, leading to degraded performance.

Our findings regarding these claims are presented below. Please note that more claims have been made by the authors. However, owing to time constraints, we could not test all of them.

METHODOLOGY

Random Latent Exploration

The new idea presented by Mahankali et al. [1] is to augment the reward function by adding a randomized term that incentivizes the agent to explore a larger portion of the state space. In particular, this term is denoted as $F(s, z)$, or the intrinsic reward function, where $s \in S$ is any state of the state space S, and $z \in R^d$ is a d-dimensional vector sampled from a given distribution P_z.

If, at time step $t \in \{0,1,2,\ldots,T\}$, the agent in state s_t takes action $at \sim \pi(\cdot|s_t)$ from policy π, it obtains the reward $r(s_t,a_t)$. In RLE, we train a latent-conditioned policy network $\pi(\cdot|s,z)$ and a latent-conditioned value network $V_\pi(s,z)$ to estimate and maximize the expected sum of discounted rewards from a given state, taking into account the random term z:

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

In the above equation, γ describes the denotes factor, and $F(s_{t+1}, z) = \varphi(s) \cdot z$, where φ: $(st+1)$: $S \rightarrow R^d$ is a feature extraction network. The feature network is updated as a linear combination of the weights of the previous feature network and those of the value network. Both φ and z are d-dimensional vectors.

In the pseudocode algorithm, in the mathematical formulation, the authors state that the feature network is updated as a linear combination of the weights of the previous feature network and policy network. We contacted the authors, and they confirmed that it should read $\phi \leftarrow \tau \cdot V^{\pi} + (1 - \tau) \cdot \phi$ (i.e., the value network should be used, not the policy network). However, they stated that it would be interesting to explore which network should be used to update a feature network.

Although the value equation is presented in the study by Mahankali et al. [1], in their implementation the authors introduced intrinsic and extrinsic reward coefficients. In the experiments, they set the intrinsic reward coefficient $\ll$ the extrinsic reward coefficient; thus, they did not directly optimize the value equation. Furthermore, this study stated that z "is resampled at the start of each trajectory." However, in the pseudocode, they note that z is resampled at the end of each trajectory or after a certain number of steps. Additionally, this study implemented the latter approach, we adopted the same approach for our experiments.

Mahankali et al. [1] split the critic's head into two: one head predicts the intrinsic value function, and the other predicts the extrinsic value function. Figure 1 depicts how the action logits, intrinsic value, and extrinsic value for a given observation and z are calculated. Note the residual connection from the first to the second element-wise addition. This residual connection is not mentioned in the paper but is present in the code provided for the Atari environment.

To continue the example from the introduction, in Figure 2, the agent would prioritize left, as this promises a greater reward. It is not hard to imagine that for higher-dimensional environments, every time we start a new trajectory and correspondingly sample a new z, the agent will explore a different region of the state space; thus, we can discover non-obvious paths to maximize the rewards.

Hyperparameters

Wherever possible, we used the same hyperparameters as the authors, except for the z-resampling frequency in Atari games. For the parameters that were missing a description in their work [1], we adopted values that seemed reasonable.

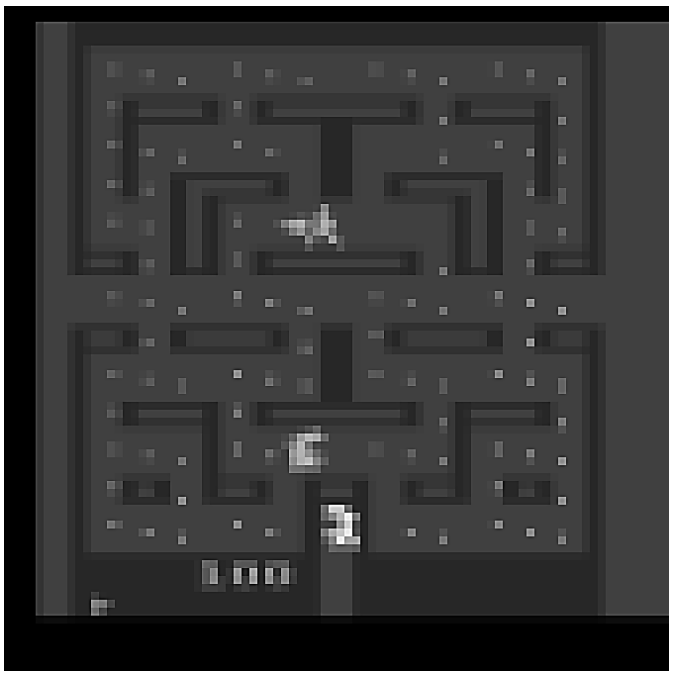

Figure 1. The Alien environment is an example of the Atari games, consisting of a maze filled with 'eggs' to destroy while being hunted by aliens. A flamethrower or occasional power-up may be used to scare the aliens.

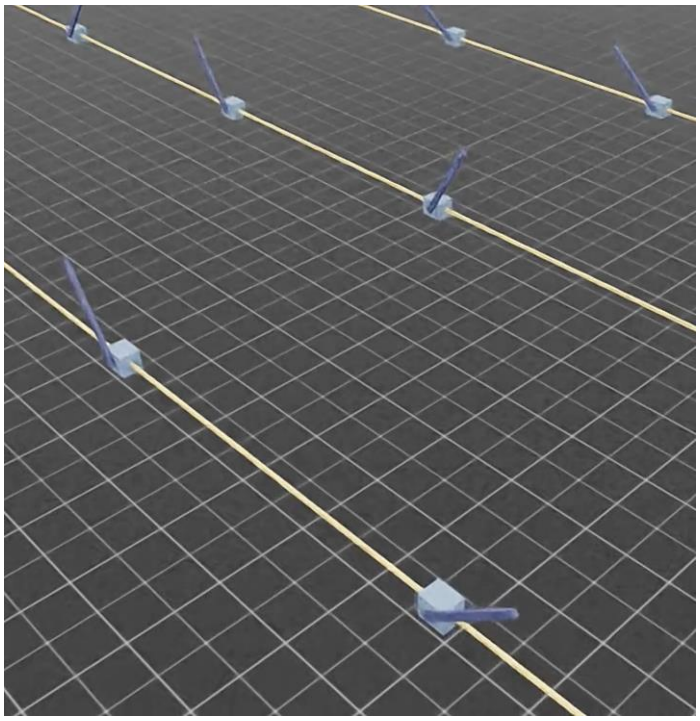

Figure 2. The CARTPOLE environment is available in Isaac Lab.

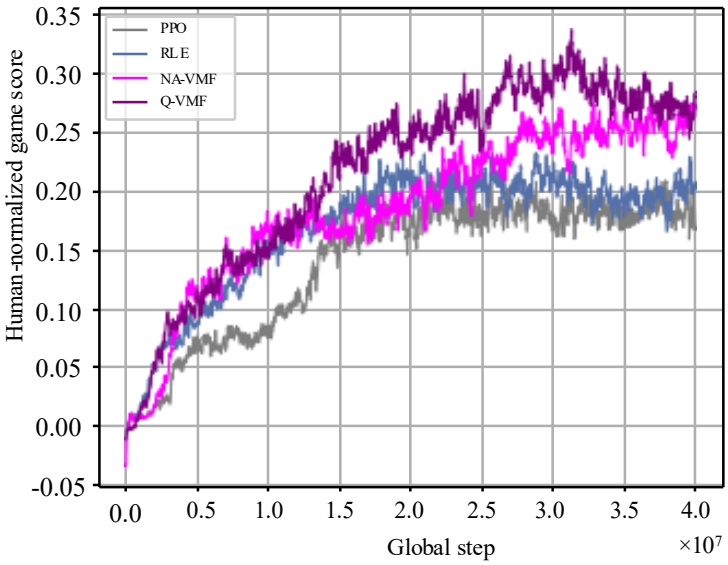

Figure 3. Game score.

Experimental Setup and Code

Four Room

Figure 3 shows the FourRoom environment, where we have 50 × 50 + 4 states that are divided into four rooms of size 25 × 25 and four holes in the walls located at positions 10 horizontally and 5 vertically, starting the count at 1 from the inner corners. Exploring the state space, beginning from the initial state marked in red, becomes a deep exploration task because the holes are only 1" pixel wide. There is also a version of the FourRoom environment that has a goal. The agent receives a +1 reward when it reaches the goal state, which is denoted in green. At every step, the agent can move one step vertically or horizontally; however, if it runs into a wall, it remains in place [1]. Sutton et al. (1999) [5] introduced the FourRoom environment and used action stochasticity. We asked Mr. Mahankali whether they also used action stochasticity, and he stated that they did not use it in their experiments.

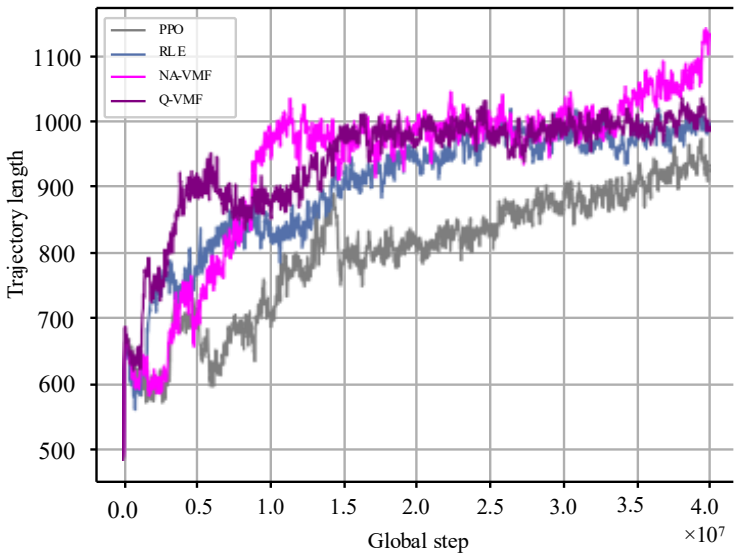

Figure 4. Trajectory length.

The study did not provide any code for the FourRoom environment; therefore, we implemented this environment using the Gymnasium library introduced by Towers et al. (2024) [6]. The FourRoom environment was implemented so that a flag could be passed to the environment constructor to choose whether the environment is reward-free or has the goal in the bottom-left corner. We adapted the PPO, NoisyNet, and RLE codes from the Atari environment, whereas the RND code was adapted from the Isaac environment. A wrapper was also implemented to record state visitation counts. Before adapting the RLE code from the existing Atari implementation, we first implemented it based on the description in the study by Mahankali et al. (2024) [1]. However, the paper lacked several details present in the Atari code (e.g., split critic's head and residual connection); therefore, we decided to proceed with the adapted code to obtain consistent results across different environments.

First, we investigated how RLE performs in a reward-free setting in comparison to PPO, NoisyNet, and RND. Mahankali et al. (2024) [1] claimed that RLE exhibits strong explorative behavior, based on a single heatmap of state visitation counts. To assess whether this result was expected or exceptional, we trained each algorithm 20 times. Because it is not reasonable to compare 20 state visitation heatmaps, we used the Shannon entropy, introduced by Shannon (1948) [7], of the state visitation counts as a proxy for how well an agent explored the environment. With increasing entropy, the distribution of state visitation counts becomes more uniform. Thus, the higher the entropy, the better the explorative behavior is.

We also ran RLE, PPO, NoisyNet, and RND in the FourRoom environment to observe how RLE compared with the other algorithms in an environment with a reward. Similar to the reward-free setting, we run each algorithm 20 times on different seeds. However, instead of assessing the performance of the algorithms using Shannon entropy, we used an average game score. The game score was the average undiscounted return over the last 128 episodes. The average game score for a certain algorithm is the average of the game scores of the different runs for this algorithm.

Furthermore, we explored whether the intrinsic reward function guided the generation of diverse trajectories in a reward-free environment. To assess this, the intrinsic reward function for the four latent vectors z and four trajectories generated using the same latent vector **z** were saved at the end of each experiment.

We also tested the degree to which the RLE is indifferent to the distribution of the latent vector. To test this, we ran RLE with different latent vector distributions. The following distributions were

considered: standard normal distribution, standard uniform distribution, von Mises distribution with $\mu = 0$ and $\kappa = 0.3$, and exponential distribution with $\lambda = 0.3$. We ran 20 experiments for each latent vector distribution and recorded the state-visitation entropy. The same hyperparameters were used as previously described. Note that all latent vectors were normalized using the L2-norm so that all latent vectors were on a unit sphere around the origin. This ensures that all intrinsic rewards are within the interval [-1, 1]. Mahankali et al. (2024) [1] performed a similar ablation study in an Atari environment. However, they did not normalize the latent vectors for all the distributions. They only applied normalization to the normal distribution, which they then denoted as "spheres". They then compared the "sphere" to the unnormalized uniform distribution and the unnormalized normal distribution. We normalized all distributions because the extrinsic and intrinsic reward coefficient hyperparameters already control the weighting between the intrinsic and extrinsic rewards; therefore, we wanted the intrinsic rewards to be in the interval [-1, 1].

Atari

The study investigated and chose the Atari environment, introduced by Mnih et al. (2013) [8], as a discrete action-space deep-RL benchmark. This environment comprised 57 different games, among which we performed experiments on two games, namely Alien and Stargunner. To understand the code provided by the authors, we relied mostly on official environmental documentation and relevant studies. The agent perceives the four most recent 84 84 grayscale frames. An example is shown in Figure 1. During training, every trajectory (episode) always commences from the same initial state and can be terminated in one of two ways: a game event occurs, or the maximum episode length is reached. The former refers to the loss of a 'life' or a task being completed. Because the authors provided the code for this environment, we did not modify it significantly. The main claims that we tested were the superior performance of the RLE and deeper exploration.

The score S_G that we use to compare the different algorithms is normalized as described in Badia et al. (2020) [9]. After considering the score that an average human achieves S_H, as well as the score an agent achieves by always choosing a random action S_R, which gives us the human normalized score $S_n = \frac{S_G - S_R}{S_H - S_R}$. According to Badia et al. (2020) [9], in the case of Alien, we have $S_H = 7127.7$ and $S_R = 227.8$, and for Stargunner, we have $S_H = 10250$ and $S_R = 664$. With normalization, we can obtain a better comparison of how much the agent behaves like a random agent ($S_N = 0$), like a human player ($S_N = 1$), or even better, $S_N > 1$. Moreover, normalization allows for comparisons between games, adjusted for the general difficulty of a game.

Each game was run with all four algorithms (RLE, PPO, NoisyNet, and RND), and the most important comparative measure to evaluate the agent's performance is the game score, which was computed from the extrinsic rewards. We also recorded the time evolution of Shannon entropy of the action distribution for each algorithm. Higher entropy indicates that the agent's selection of actions out of all available actions is broader and less concentrated, hinting at more thorough exploration capabilities.

Isaac Lab

The code provided in the study by Mahankali et al. (2024) [1] was developed for Isaac Gym, which was deprecated. Therefore, we decided to reimplement it in Isaac Lab, which is a reinforcement learning framework built on top of the Isaac Sim. However, the migration from Isaac Gym to Isaac Lab is non-trivial because the software has been significantly changed. For example, Isaac Lab uses a more advanced physics engine with various asset extensions.

We base our implementation on the example provided in the study by Serrano-Muñoz et al. (2022) [10]. Using their implementation for PPO as an example, we identified the main structure for developing our own version. The Environment file (Cartpolerleenv.py) extends the basic CartPole environment class with the components necessary for RLE. This includes, for instance, latent vector generation and intrinsic and extrinsic rewards. The Configuration file (ppocfgrle.yaml) contains the hyperparameters

for the training process. The model.py file contained the policy and value network implementation for the RLE case. The rle_network contains the feature network implementation for latent vector generation. Besides these files, we implemented the Trainer file (trainrle.py), which handles the arguments, acts as an entry point, initializes Isaac Sim, sets up the tracking, configures the environment, and initializes the runner (runnerrle.py), which instantiates the agent, coordinates the interactions between the agent and the environment, and executes the training. Finally, we have the PPORLE algorithm (pporlesk.py), is a custom PPO that handles the dual rewards for the two heads of the value network, the generalized advantage estimation, and the learning scheduling, among others.

For the development of each part, we did not rely on the authors' code unless we checked the default parameters when they were not stated in the study [1], and as a double check of the logic of the optimization, because the complexity of the environment interaction with Isaac Lab is higher and is mostly managed by almost completely modifying the SKRL base PPO code. Because this implementation is highly structured, we had to implement workarounds. For example, this algorithm has implemented the actor-critic structure handling, but not a feature network. Therefore, we handle this by generating a latent vector in the environment, computing the intrinsic rewards, and passing it through the runner to the modified PPO algorithm to handle the dual rewards and the soft update of the feature network during policy updates and environment resets. In addition, the maximum number of steps was approximately 299 per episode in the implementation of SKRL base libraries. However, Mahankali et al. (2024) [1] used a maximum step number of 500. Because increasing the number of steps would require a higher number of computations, we opted to keep the 300 steps. This means that our results are a scaled-down version of those of the authors.

Queue-Based von Mises-Fisher

Mahankali et al. (2024) [1] suggested that it might be worthwhile to explore the use of a latent vector distribution that changes during training. Our hypothesis is that, at the start of the training, the latent vector distribution P_z should provide diverse z because the agent should explore the environment. However, with increasing training progress, the agent should emphasize exploration. We use the von Mises-Fisher (vMF) distribution with the parameters, the mean direction, and κ, the concentration, to model P_z. The goal is to adjust gradually during training so that it converges to a point in the space where P_z generates z samples that consistently produce high returns. Furthermore, κ should be gradually increased during training to increase the probability of sample z being close to the mean direction. Because PyTorch does not provide an implementation to sample from this distribution, we used the implementation described in the study by Pinzón and Jung (2023) [11], which is based on rejection sampling. We maintained a queue of z values with good episodic returns and a queue with the respective returns. After every finished episode, the queues are updated if necessary. We then update as a linear combination of z values in the successful memory. Weighting is performed using the returns in the return memory. κ should be large if it converges to a value such that the corresponding distribution reliably yields z values with high returns. Thus, κ is updated by linear interpolation between 0 and 30. The weight is the average pairwise cosine similarity between the $\mathbf{z}$ values in the success memory.

Neural-Adaptive von Mises-Fisher

We further explore the idea of using a distribution that allows for natural manipulation of the exploration-exploitation trade-off, allowing for direct directional and density κ control. Given that the steps of an episode have temporal dependency, we opted to process the trajectories with an long short-term memory (LSTM) that encodes the short- and long-term patterns observed in the generated data. When a new episode ends, the hidden state is reset because between episodes, the agent should not retain any information; however, the actual knowledge encoded in the weights of the LSTM is preserved.

After the LSTM preprocesses the state features generated by the agent network, the μ-network receives these processed features to predict a good direction of exploration. However, the direction for

each state can be understood using different degrees of confidence. The κ-network then predicts the corresponding confidence, κ value, for the predicted condition of each state. Once κ and are obtained, we can sample from the vMF distribution to obtain the suggested direction for the next action to be taken. The sampling allows for a degree of exploration that cannot be represented by a deterministic approach, since even with a high κ concentration, we can sample slightly different directions from, which can change the results with respect to a z equal to.

The obtained z, in conjunction with the features, is fed into the reward network that acts as a substitute for the dot product in the RLE network. Therefore, the reward network operates with the respective state features and z-vector to produce an intrinsic reward to guide exploration.

Computational Requirements

All FourRoom experiments were performed on an Apple M2 Pro with 16 GB RAM. Every experiment was conducted in a reward-free environment and an environment with a goal. Every algorithm and every RLE variant with different latent vector distributions was run 20 times with different seeds in each of the two environments. This resulted in a total of 280 runs. It took approximately 1 min to perform a single PPO run, approximately 1 min and 20 s to perform a single NoisyNet run, approximately 2 min to perform a single RLE run, and approximately 2 min and 20 s to perform a single RND run. The experiments for the different algorithms required a total time of approximately 4.4 hours. In addition, we trained the RLE using three different latent vector distributions (apart from the standard normal distribution). These experiments took approximately 4 h. Thus, the total computing time for all FourRoom experiments was approximately 8.4 hours.

All the experiments in the Atari environment were conducted using Google Colab. We experimented with the central processing unit (CPU), T4 graphics processing unit (GPU), and tensor processing unit (TPU) v2-8, and used the Weights & Biases (W&B) for real-time visualization of the game scores and other measures. Finally, we chose the GPU for all runs. Overall, the W&B workspace now contains 56 runs with a total computation time of approximately 241.2 hours, of which eight runs were used for the results of this project. Of that total, approximately 76.7 compute hours were spent on Stargunner, and the remaining 164.5 compute hours were spent on Alien.

The Isaac Lab experiments were performed on an Asus the ultimate force (TUF) Gaming A17 with an RTX 3060 Laptop GPU. The GPU characteristics are below the suggested minimum requirements for the video random access memory (VRAM) memory of NVIDIA; however, for developing the experiments for the CARTPOLE environment, it was sufficient. The compute unified device architecture (CUDA) version used was 11.8.

However, we would like to note that Mahankali et al. (2024) [1] had the HPC resources of the MIT Supercloud and Lincoln Laboratory Supercomputing Center. Within the framework of this course, we did not have the equivalent computing power to replicate all experiments.

RESULTS

Results Reproduced by Mahankali et al. (2024) [1]

FourRoom

The results are from the experiment in which each algorithm was run 20 times in a reward-free environment. RLE exhibited the best performance, followed by RND, NoisyNet, and PPO. Furthermore, RLE had the narrowest confidence interval. Although Mahankali et al. (2024) [1] did not perform an equivalent experiment, they concluded from the state visitation heatmap that RLE showed the best explorative behavior in this setting. This aligns with the results of the present study.

The results of the experiment show that every algorithm was run 20 times in an environment with a goal. Mahankali et al. (2024) [1] trained five seeds in an environment with a goal for each algorithm and found that the average score for RLE and NoisyNet was 0.6, while the average score for RND and

PPO was 0. In our experiment, RLE achieved the highest game score by a large margin, whereas NoisyNet and RND performed similarly poorly, with an average game score of <0.2 at the end of the training. The PPO achieved an average game score of zero at the end of training. Thus, the result that RLE is among the best-performing algorithms in this setting is confirmed by our experiment.

EVALUATION OF LATENT VECTOR DISTRIBUTIONS IN RLE

An experiment was conducted to compare the different RLE variants in a reward-free environment. Each variant was executed 20 times using distinct latent vector distributions. The variants employing standard normal and von Mises distributions demonstrated superior performance, significantly outperforming those employing standard uniform and exponential distributions. Interestingly, the performance of the uniform and exponential variants was comparable to that of the RND. These findings suggest that the choice of the latent vector distribution plays a critical role in exploration quality. Although the von Mises distribution was parameterized to flatten its peak, making it resemble a uniform distribution, it still exhibited a performance similar to the standard normal variant, indicating the nuanced influences of distribution shape on learning behavior.

A similar experiment was conducted in an environment with a defined goal, running each RLE variant 20 times. The von Mises variant achieved the highest average game score, followed by the standard normal, exponential, and uniform variants. The performance gap was considerable: the lowest performing variant achieved an average score just above 0.2, whereas the best variant approached a score of 0.8. The variant ranking mirrored that observed in the reward-free environment, reinforcing the conclusion that the latent vector distribution has a notable impact on RLE performance.

To further examine the agent's behavior, state visitation heatmaps were generated for each algorithm and RLE variant in both environments. For the reward-free environment, the run with the highest entropy was selected per variant. For the goal-oriented environment, the run with the highest game score was used. In a reward-free setting, RLE and NoisyNet demonstrated comprehensive exploration, covering all four rooms. In contrast, the PPO and RND left substantial areas unexplored. In the goal-directed setting, RLE, NoisyNet, and RND concentrated visits along paths from the start to the goal, whereas PPO remained confined to a single room. Among the RLE variants, those with standard normal, von Mises, and exponential distributions thoroughly explored all four rooms. However, the standard uniform variant fails to cover the entire state space. Despite this, all variants frequently traverse paths that approximate the shortest route to the goal. The standard normal and von Mises variants showed the broadest exploration, while the uniform and exponential variants left large areas unvisited.

Finally, the trajectory analysis revealed that the intrinsic reward mechanism effectively encouraged diverse explorations. The agent trajectories tended to converge in the regions where the intrinsic reward values were the highest. All reported 95% confidence intervals were computed using bootstrapping.

Atari

After running each game for a total of 40-million-time steps, we obtained results.

For the Alien game, as shown in Table 1, the algorithm performs similarly to the results of the authors. Only NoisyNet performed much worse in our run, obtaining a score of 648, while the authors reported a score of approximately 1113. However, regarding *claim*, we can confirm that RLE obtains the highest score, which is evident from the plot of human normalized scores. There is a correlation between the game score and the entropy bonus. As noted, the entropy bonus is added to the PPO loss to enhance the agent's exploratory behavior. One could assume that RLE has the highest entropy bonus, indicating that it shows the best explorative behavior.

In the case of the Stargunner game, as shown in Table 2, the RLE algorithm does not yield the best game score, whereas PPO does. This result strongly differs from the findings of the present study.

Table 1. The alien game.

Algorithm	Game Score	Human-Norm
RLE	1626	0.203
PPO	1384	0.168
NoisyNet	648	0.061
RND	1048	0.119

Table 2. The Stargunner game.

Algorithm	Game score	Human-Norm
RLE	42481	4.362
PPO	54564	5.623
NoisyNet	27530	2.803
RND	21565	2.180

In more detail, we find that RLE obtains a game score of around 42481, while the authors report 64011. In addition, their implementation of NoisyNet performed much better at 42645, while our agent only got 27530 in the end. Thus, in this case, we have to object to claim (1) that RLE has the largest entropy bonus. This indicates that RLE samples actions with the greatest variety, potentially reflecting diverse behaviors.

Isaac Lab

The results from our Isaac Lab replication are similar to Mahankali et al [1] RLE, which shows the most stable performance in the CARTPOLE environment. The difference in the maximum score of Mahankali et al. (2024) [1] comes from the maximum length of the episodes, which is maintained at 299 steps in our replication.

Results Beyond Mahankali et al. (2024) [1]

Atari

One interesting observation that we made during our experiments is that in Alien, the RLE agent seems to obtain lower rewards on average compared to the baselines but still has the highest overall game score. We conjecture that this means the agent with RLE behaves more cautiously, trying to stay alive for longer and collect lower/safer rewards, but over a longer time horizon, meaning that the overall accumulated reward (game score) still turns out to be superior compared to the other algorithms. This was confirmed by the RLE, which had the largest trajectory (episode) length. However, this is probably not the entire explanation, because in Stargunner, RLE also receives a very low average reward, which cannot be offset simply by having the longest trajectory length. Moreover, given that we also normalized the game scores, the result suggests that the agent is better at playing Stargunner and worse at playing Alien than an average human.

Furthermore, as mentioned in the hyperparameter overview, we increased RLE's **z**-sampling frequency of the RLE from every 1280 time steps (authors) to every 500, but we did not find materially different results. Another observation we made is that, for some reason in the RLE implementation of Alien, the results differ drastically between using a GPU and a CPU/TPU for the computations, but the underlying cause of this is beyond our understanding.

Adaptive von Mises-Fisher

The results from adaptive vMF are promising Figure 3. Furthermore, Figure 4 shows that NA-vMF has the longest trajectory lengths among the four algorithms. QvMF in Figure 4 has trajectory lengths similar to those of PPO at the end of the training. Both our adaptive vMF implementations outperform the other algorithms for Alien, suggesting that it would be worthwhile to explore this approach further for other Atari games and continuous action spaces.

Furthermore, we maintain the same resampling logic as in the RLE. However, for the neural-adaptive vMF, the loss terms can naturally provide the agent with the feedback needed to resample **z**'s in the appropriate moment. Thus, resampling **z** in fixed intervals is not necessary. However, this was not investigated in the present study.

DISCUSSION

Most of our experiments demonstrated that RLE outperformed the baselines in the FourRoom and Isaac Lab environments. We obtained mixed results only for the Atari. Although RLE outperformed the baselines in Alien, it did not outperform PPO in Stargunner. Owing to computational and time constraints, we could only perform experiments on two out of 57 Atari games and one out of nine Isaac Lab environments.

Additionally, in the fourRoom experiments, we could see that RLE explored a larger portion of the state space than the baselines. From the diverse trajectories generated by RLE, we conclude that RLE is able to produce diverse behaviors. In the two Atari environments, Alien and Stargunner, the action entropy of RLE was higher than that of the baselines. This suggests that the RLE exhibits a more diverse behavior than the baselines.

Moreover, this study investigated using FourRoom experiments. In our experiments, the RLE generated diverse trajectories such that the agent could explore all four rooms.

Nonetheless, this study did not supported the performance of RLE with the standard normal and von Mises latent vector distributions was better than that with the standard uniform and exponential latent vector distributions. However, Mahankali et al. (2024) [1] conducted experiments with different latent vector distributions in Atari environments. They reported that RLE was robust with respect to the latent vector distribution.

In our replication of the CARTPOLE experiment, we observed that PPO with random rewards performed worse than RLE, where the random rewards were correlated with the states. Thus, we can confirm finding 5 reported by Mahankali et al. (2024) [1].

Furthermore, Mahankali et al. (2024) [1] concluded that conditioning the policy network on a latent vector is important. However, in this figure, RLE without latent vector conditioning shows similar performance to RLE with latent vector conditioning in three out of the four Atari games. In only one game, the latent vector-conditioned RLE version performed better. Therefore, the importance of latent vector conditioning should be investigated further.

We have also considered the idea of Mahankali et al. (2024) [1] to change the latent vector distribution over time by implementing queue-based adaptive vMF and neural-adaptive vMF RLE variants. Both variants outperformed the standard RLE in the Atari game, Alien. It is worthwhile exploring how these RLE variants perform in other Atari games and various Isaac Lab environments.

What Was Easy

The main concept of this study is straightforward and easy to understand. Moreover, the code provided by the authors for the Atari environment was ready to be run.

What Was Difficult

The greatest challenges were the lack of computational resources and time limitations. Moreover, this study lacks a proper description of the RLE architecture.

Communication With Original Authors

During our replication study, a number of questions, mainly regarding parameters and model/environment architecture, came up, which we were unable to answer by ourselves; therefore, we

sent an email to Mr. Mahankali, and received responses a few days later. Our questions and Mr. Mahankali's answers are provided upon request.

CONCLUSION

From our experiments, it became evident that RLE is a suitable alternative to RND and NoisyNet in environments where deep exploration is required. However, because of limited computational resources and time constraints, we could not test all of the claims reported by Mahankali et al. (2024) [1]; however, our findings are broadly consistent with their results. To strengthen these findings, it would be worthwhile examining RLE in the full range of Atari games and Isaac environments. Furthermore, because the adaptive vMF techniques showed promising results in the Alien game, they should be evaluated in other games and in the continuous action-space domain.

REFERENCES

1. Mahankali S, Hong ZW, Sekhari A, Rakhlin A, Agrawal P. Random latent exploration for deep reinforcement learning. arXiv. [preprint]. 2024. arXiv:2407.13755. doi:10.48550/arXiv.2407.13755.
2. Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O. Proximal Policy Optimization Algorithms. arXiv [Preprint]. 2017. arXiv:1707.06347. doi:10.48550/arXiv.1707.06347.
3. Fortunato M, Azar MG, Piot B, Menick J, Osband I, Graves A, et al. Noisy Networks for Exploration. arXiv [Preprint]. 2017 Jun 30. arXiv:1706.10295. doi:10.48550/arXiv.1706.10295.
4. Burda Y, Edwards H, Storkey A, Klimov O. Exploration by Random Network Distillation. arXiv [Preprint]. 2018 Oct 30. arXiv:1810.12894. doi:10.48550/arXiv.1810.12894.
5. Sutton RS, Precup D, Singh S. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. Artif Intell. 1999;112:181-211. doi:10.1016/S0004-3702(99)00052-1.
6. Towers M, Kwiatkowski A, Terry J, Balis JU, De Cola G, Deleu T, et al. Gymnasium: A Standard Interface for Reinforcement Learning Environments. arXiv [Preprint]. 2024. arXiv:2407.17032. doi:10.48550/arXiv.2407.17032.
7. Shannon CE. A mathematical theory of communication. Bell Syst Tech J. 1948;27:379-423. doi:10.1002/j.1538-7305.1948.tb01338.x.
8. Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, et al. Playing Atari with Deep Reinforcement Learning. arXiv [Preprint]. 2013. arXiv:1312.5602. doi:10.48550/arXiv.1312.5602.
9. Badia AP, Piot B, Kapturowski S, Sprechmann P, Vitvitskyi A, Guo ZD, et al. Agent57: Outperforming the Atari human benchmark. In: Daumé III H, Singh A, editors. Proceedings of the 37th International Conference on Machine Learning. 2020 Jul 13-18; Vienna, Austria. Vol. 119. Proceedings of Machine Learning Research. PMLR; 2020. p. 507–17. Available from: https://proceedings.mlr.press/v119/badia20a.html.
10. Serrano-Muñoz A, Chrysostomou D, Bøgh S, Arana-Arexolaleiba N. skrl: Modular and flexible library for reinforcement learning. arXiv [Preprint]. 2022. arXiv:2202.03825. doi:10.48550/arXiv.2202.03825.
11. Pinzón C, Jung K. (2023). Fast Python sampler for the von Mises Fisher distribution [Online]. HAL Open Science. Available from: https://hal.science/hal-04004568v3.

