Improving PPO Sample Efficiency on Sparse Reward Environments

Johny Tarbouch & Qasim Sefaldeen
https://github.com/johnnyTar/PPO-Sample-Efficiency/tree/main

Abstract

This project investigates the combination of Proximal Policy Optimization (PPO) with Self-Imitation Learning (SIL) and Random Network Distillation (RND) to address sample efficiency challenges in sparse reward reinforcement learning environments. We implement and evaluate a hybrid approach that maintains PPO's stability while leveraging SIL's ability to learn from past successful experiences through prioritized replay and RND to improve exploration. Our experiments on MiniGrid environments, particularly the DoorKey task, demonstrate that PPO+SIL achieves improved sample efficiency and higher success rates compared to vanilla PPO. The approach uses an adaptive success threshold mechanism and prioritized experience replay to efficiently exploit rare positive experiences in sparse reward settings. However, we found that RND did not provide noticeable improvements when combined with PPO in this setting.

1 Introduction

Reinforcement learning in sparse reward environments presents challenges for exploration and sample efficiency. On-policy methods like PPO excel at stable policy updates, however, are sample-inefficient, particularly in sparse-reward environments such as MiniGrid. Off-policy methods can reuse past experiences but suffer from instability. SIL and RND address these challenges in different ways: SIL enables agents to learn from past high-reward decisions, encouraging consistent reproduction of successful behaviors, while RND improves exploration. Our theory is that combining on/off-policy methods can improve the sample inefficiency problem of PPO.

This leads to our research question: *Can the integration of experience replay buffers or intrinsic reward mechanisms improve the sample efficiency of PPO in sparse-reward environments such as MiniGrid?*

The contributions include: (1) a practical implementation combining PPO with SIL using prioritized experience replay, (2) adaptive success threshold mechanisms for filtering good experiences, (3) evaluation on MiniGrid environments demonstrating improved sample efficiency, and (4) analysis of trade-offs between exploration and exploitation in hybrid on-/off-policy approaches. RND was incorporated to encourage exploration, though it did not yield noticeable gains.

2 Related Work

Proximal Policy Optimization was introduced by Schulman et al. [2017] as a policy gradient method that addresses the stability issues of earlier approaches. PPO's clipped surrogate objective prevents large policy updates while enabling multiple epochs of learning from collected data.

However, prior work by Chan has shown that PPO performs better in dense reward environments (Cart pole) compared to sparse reward (MiniGrid) environments.

Therefore, **Self-Imitation Learning**[Oh et al., 2018] proposes learning from past good decisions by maintaining a buffer of high-return experiences for additional policy updates, to improve sample efficiency problems, and has shown improvement over PPO. Similarly, the **MineDojo** framework Fan et al. [2022] integrates SIL with PPO to mitigate sample inefficiency in a sparse-reward environment (Minecraft) by storing high-return trajectories and reinforcing successful past experiences. **R3**[Li et al., 2024] extends PPO for discrete action spaces by replaying high-reward trajectories with clipped importance sampling ratios, yielding significant sample-efficiency gains in sparse-reward MiniGrid.

Experience Replay methods have been studied for improving sample efficiency. Prioritized Experience Replay [Schaul et al., 2015] samples transitions based on their temporal-difference error.

Random Network Distillation [Burda et al., 2018] rewards agents for visiting novel states. It uses a fixed random target network and a trainable predictor; high prediction error signals novelty and yields intrinsic exploration reward.

3 Approach

Our approach extends standard PPO with SIL that learns from past successful experiences. The core idea is to maintain both the standard PPO rollout buffer and an additional SIL buffer that stores successful episodes for replay. In addition, we incorporate RND with PPO, encouraging the agent to explore novel states by providing an exploration bonus based on prediction error.

3.1 PPO Baseline

PPO Agent: We implement a standard PPO agent with an actor-critic architecture using shared network layers and the clipped surrogate objective:

$$L^{CLIP}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right] \tag{1}$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$.

Total loss:

$$L(\theta) = L^{CLIP}(\theta) + c_1(V_\theta(s_t) - V_{target})^2 - c_2 \sum_a \pi_\theta(a|s_t) \log \pi_\theta(a|s_t) \tag{2}$$

PPO Buffer: Computes advantages using GAE: $\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t)$
$\hat{A}_t^{GAE} = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}$

Algorithm: (1) Collect trajectories, (2) Compute normalized GAE advantages, (3) Update policy via minibatch gradient ascent.

3.2 SIL Buffer with Prioritized Replay

The SIL buffer stores successful episodes and samples them for additional training. Our implementation includes several key components:

Adaptive Success Filtering: Episodes are stored in the SIL buffer only if they meet a success threshold that adapts during training. Initially, any episode with a positive return above the threshold qualifies. The threshold increases as the agent to maintain high-quality experiences and decreases when fewer than 5 successful episodes are stored to prevent deadlock.

Prioritized Sampling: Transitions are stored with priorities proportional to their returns raised to power α:

$$p_i = \max(return_i, \epsilon)^\alpha \tag{3}$$

where ϵ ensures non-zero priorities. During SIL updates, only transitions with positive advantages $(R - V(s))^+ > 0$ are used for learning.

Reward Shaping: For successful episodes, we apply reward shaping with progress bonuses that decay linearly over episode length:

$$r'_t = r_t + \frac{episode\_length - t}{episode\_length} \times 0.1 + 0.01 \tag{4}$$

making shorter, more efficient solutions, meaning more attractive during learning.

Algorithm 1 PPO with Self-Imitation Learning

Require: Environment env, SIL buffer B_{SIL}
  Initialize policy π_θ and value function V_ϕ
  for iteration $i = 1, 2, \ldots$ **do**
    Collect rollout τ using π_θ
    Update π_θ, V_ϕ using PPO on τ
    Store successful episodes from τ in B_{SIL}
    if $|B_{SIL}| \geq$ batch_size and i % update_freq $= 0$ **then**
      Sample batch from B_{SIL} with prioritized replay
      Compute advantages: $A = (R - V_\phi(s))^+$
      Perform SIL updates on π_θ, V_ϕ using positive advantages only
    end if
  end for

3.3 SIL Update Mechanism

SIL updates gradients only on experiences with positive advantages and policy loss incorporates clipping to prevent large policy updates:

$$L_{policy}^{SIL} = -\mathbb{E}_{(s,a,R)\sim B_{SIL}} \left[\min\left(r_t(\theta) \cdot A_t^+, clip(r_t(\theta), 1-\epsilon, 1+\epsilon) \cdot A_t^+\right) \cdot w_t\right] \tag{5}$$

where:

$\mathrm{A}_t^+ = \max(0, R_t - V_\phi(s_t))(\text{positive advantages only})$
$w_t = \left(\frac{1}{N \cdot P(i)}\right)^\beta (\text{importance sampling weights})$
$P(i) = \frac{p_i}{\sum_k p_k} (\text{sampling probabilities})$

The complete SIL loss combines policy and value losses:

$$L^{SIL} = L_{policy}^{SIL} + 0.5 \cdot \mathbb{E}_{(s,R)\sim B_{SIL}} \left[(V_\phi(s) - R)^2 \cdot w_t\right] \tag{6}$$

The importance sampling weights w_t correct for sampling bias introduced by prioritized replay, where β anneals from 0.4 to 1.0 during training to gradually remove bias. Only transitions with positive advantages contribute to the policy gradient.

3.4 Random Network Distillation (RND)

To promote exploration in sparse-reward environments, we integrate RND as an intrinsic reward mechanism. RND assigns bonuses based on state novelty, encouraging visits to unexplored regions. Our setup uses the standard architecture with two multi-layer perceptrons: a fixed *target network* f (randomly initialized) and a trainable *predictor network* $\hat{f}$.

Intrinsic Reward: For state s_t, the intrinsic reward is the MSE between network outputs:

$$r_t^i = ||\hat{f}(s_t; \theta_{\hat{f}}) - f(s_t)||_2^2 \tag{7}$$

Higher errors indicate novelty, producing larger bonuses. The total reward is

$$R_t = r_t^e + \eta \cdot r_t^i \tag{8}$$

where r_t^e is the environment reward and η scales the intrinsic term.

Normalization: We normalize inputs and intrinsic rewards using running mean and standard deviation estimates to stabilize training.

Algorithm 2 PPO with RND

Require: Env env, coefficient η
Init policy π_θ, value V_ϕ, target f (fixed), predictor $\hat{f}$, normalizers N_{obs}, N_{rew}
for iteration i **do**
  Init rollout buffer $\mathcal{B}$
  for $t = 1 \dots T$ **do**
    Take a_t from π_θ, get r_t^e, s_{t+1}
    $s'_{t+1} \leftarrow N_{obs}(s_{t+1})$
    $\epsilon \leftarrow \hat{f}(s'_{t+1}) - f(s'_{t+1})^2$
    $r_t^i \leftarrow N_{rew}(\epsilon)$
    $R_t \leftarrow r_t^e + \eta \cdot r_t^i$
    Store $(s_t, a_t, R_t, V_\phi(s_t))$ in $\mathcal{B}$
  end for
  Compute GAE $\hat{A}_t$
  for K PPO epochs **do**
    Update π_θ, V_ϕ from $\mathcal{B}$
    Update $\hat{f}$ by minimizing MSE with f
  end for
end for

4 Experiments

We evaluate our approach on MiniGrid sparse-reward environments. Vanilla PPO was able to solve Empty, Lava Gap, and DoorKey-6x6 (5x5) tasks, but not DoorKey-8x8. Therefore, we focus on the DoorKey-8x8 task, which requires agents to collect a key, unlock a door, and reach a goal location to receive a reward.

4.1 Experimental Setup

Environment: MiniGrid-DoorKey-6x6-v0 and MiniGrid-DoorKey-8x8-v0, and flat observation wrapper. The training hyperparameters for PPO have been taken from Eimer et al. [2023], and those for SIL from the original paper 1, we will investigate Success threshold, and Priority exponent further.

Table 1: Experimental Hyperparameters, highlighted are the HP tuning

PPO Parameters		**SIL Parameters**		**RND Parameters**	
Learning rate	1×10^{-4}	SIL learning rate	1×10^{-4}	RND learning rate	1×10^{-4}
Discount factor (γ)	0.99	Buffer size	10,000	Intrinsic coef. (η)	0.05
GAE lambda (λ)	0.95	Batch size	32	Embedding dim	128
Clip ratio	0.14	Update ratio	4	Update epochs	1
Entropy coefficient	0.01	**Update frequency**	[3, 5]	Obs normalization	True
Value coefficient	0.5	**Success threshold**	[0.7, 0.8, 0.9]	Reward normalization	True
Rollout size	1024	Warmup episodes	50		
Batch size	64	**Priority exponent** (α)	[0.4, 0.6, 0.8]		
Hidden dimension	256	Importance sampling (β)	0.4		
Training Configuration					
Total timesteps: 1,000,000			Environment: MiniGrid		
Random seeds: 10 [0-9]			CPU: AMD Ryzen 5 5600x & Intel(R) Core(TM) i7		

Evaluation: 10 episodes measuring episode return, success rate, episode length, and total loss.

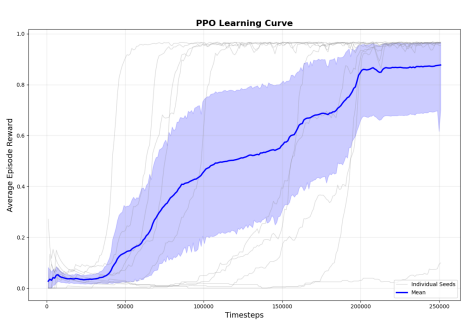
5 Results

5.1 RND

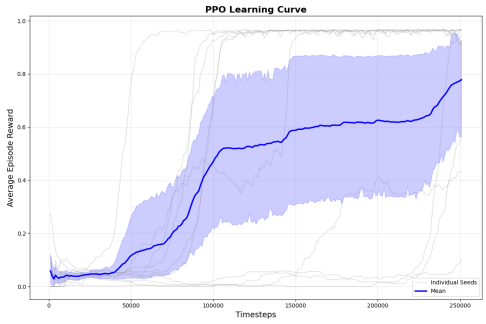
On the simpler DoorKey-6x6 task (Figure 1), the baseline PPO agent is more sample-efficient and attains a higher final average reward (0.9) than PPO+RND (0.75), with substantially lower variance.

On the more complex DoorKey-8x8 task (Figure 2b), neither method succeeds within one million timesteps. PPO shows modest learning, whereas adding RND is detrimental and yields even lower returns.

Overall, in our setup RND does not improve PPO: it increases variability on the simple task and hinders performance on the harder one.

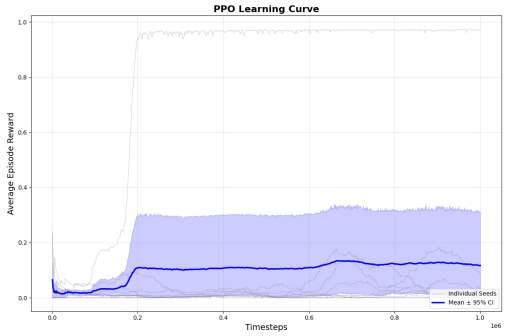


(a) PPO Baseline Learning Curve DoorKey-6x6.

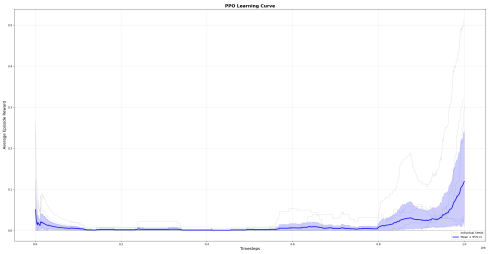


(b) PPO with RND Learning Curve DoorKey-6x6.

Figure 1: The solid line represents the mean average reward, and the shaded area is the confidence interval.



(a) PPO Baseline Learning Curve DoorKey-8x8.



(b) PPO with RND Learning Curve DoorKey-8x8.

Figure 2: Both approaches struggle to learn effectively, and incorporating RND further degrades performance relative to the baseline.

5.2 SIL

Our experiments demonstrate clear benefits of adding SIL to PPO on different HP 1:

Sample Efficiency: PPO+SIL (Figure 3b) achieves 50% higher reward and success rates compared to vanilla PPO within the same number of training steps. The hybrid approach reaches stable performance in 9 different seeds, suggesting more reliable performance.

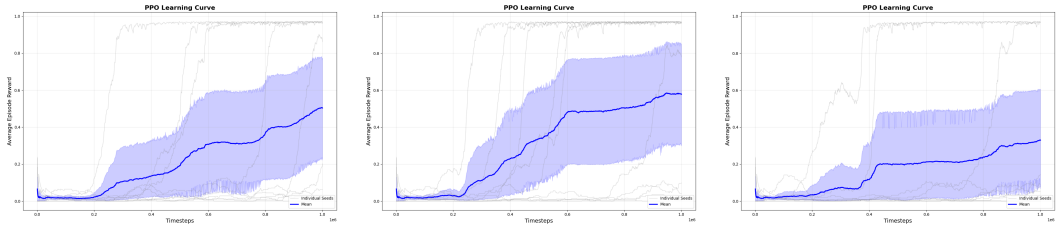
Success threshold: Episodes are retained in the SIL buffer only if their returns exceed this threshold (Figure 3). We found that, lower thresholds (0.7) include more diverse but potentially suboptimal trajectories, while higher thresholds (e.g., 0.9) ensure quality but may be too restrictive in sparse reward environments where successful experiences are rare.

Update frequency: Determines after how many PPO updates a SIL update should occur. Training on frequencies (3 and 5) shows that lower update frequencies lead to slower training and hinder performance compared to 5, because smaller frequencies make the agent biased toward suboptimal goals.

SIL Alpha (α): This parameter controls the balance between prioritized sampling and uniform sampling from the SIL buffer (Figure 4). Lower values (0.4) promote diverse experience sampling,

including suboptimal but potentially enhanced exploration. Higher values (0.8) emphasize high-quality experiences, accelerating convergence when sufficient successful episodes are available. Higher alpha outperforms smaller, indicating that SIL prefers exploitation.

The SIL buffer effectively identifies and exploits successful patterns, with analysis showing that prioritized replay focuses on key collection and door opening behaviors that form the foundation for consistent task completion.

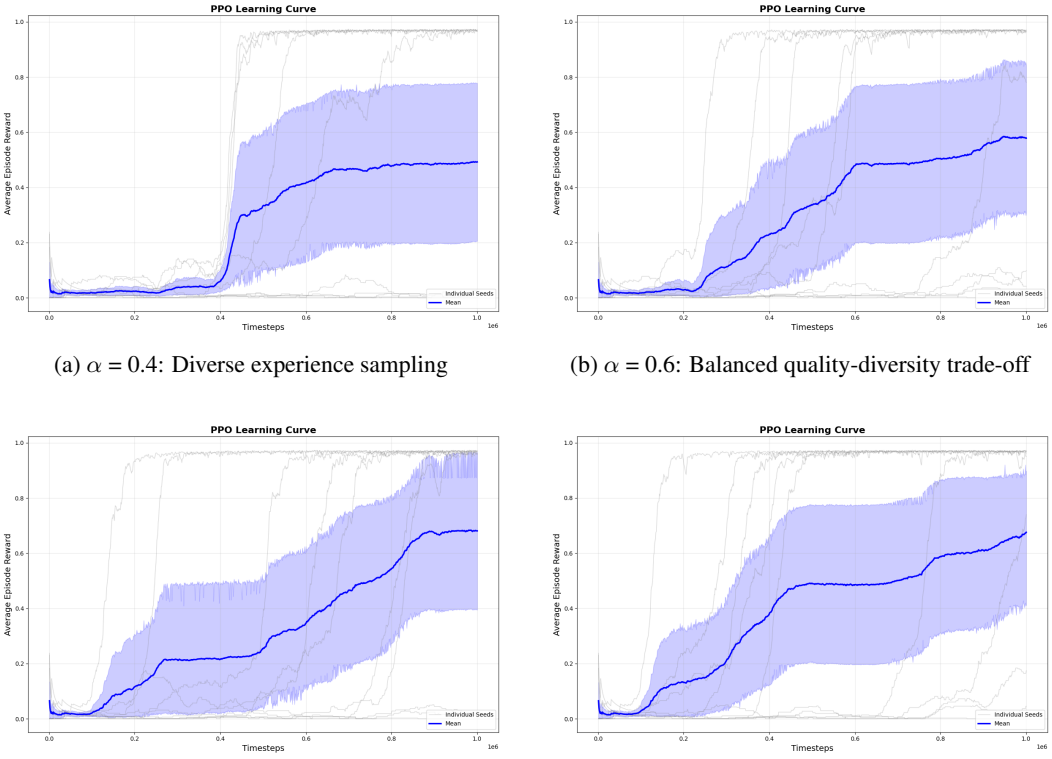


(a) Success threshold 0.7: Includes diverse experiences

(b) Success threshold 0.8: Balanced quality-diversity trade-off

(c) Success threshold 0.9: High-quality experiences only

Figure 3: PPO+SIL across different success thresholds, show how threshold selection affects sample efficiency and convergence behavior.



(a) $\alpha = 0.4$: Diverse experience sampling

(b) $\alpha = 0.6$: Balanced quality-diversity trade-off

(c) $\alpha = 0.8$: High-quality experience prioritization

(d) $\alpha = 1.0$: Pure quality-based selection

Figure 4: PPO+SIL across different alpha parameters, showing how alpha selection affects sample efficiency and convergence behavior.

6 Discussion

6.1 RND

In the DoorKey-8x8 environment, PPO+RND is highly unstable and sensitive to random seeds, yielding widely divergent outcomes. While the method may show some benefit on the smaller 6x6

task, its effectiveness breaks down as the state space grows, indicating that the exploration bonus does not scale robustly.

The core problem is that the policy becomes novelty-seeking rather than task-seeking. Because extrinsic rewards are extremely sparse, many trajectories provide no task reward, so the intrinsic RND bonus dominates the learning signal. Although the bonus should decay as states become familiar, in our setup it remains sufficient to reinforce repetitive, local behaviors that reliably produce small prediction errors. As a result, the agent gets trapped in predictable (novelty loops), neglects goal-directed exploration, and fails to solve the task.

6.2 SIL

Our results demonstrate that combining PPO with Self-Imitation Learning provides meaningful improvements in sparse reward environments by addressing inefficient use of rare successful experiences.

What Worked: The adaptive success threshold helps keep low-quality experiences from filling the buffer. Prioritized replay based on advantage estimates focuses learning on informative transitions. Integration with PPO maintains stability while adding sample efficiency benefits.

Limitations: Requires additional hyperparameter tuning and needs computational power. Performance gains may decrease when successful experiences become repetitive.

Future Work: Extensions include adaptive prioritization schemes, integration with curiosity-driven exploration. Further investigate performance when successful experiences become repetitive.

The work demonstrates that hybrid approaches combining on-policy stability with off-policy sample efficiency represent a promising direction.

Our research question is answered affirmatively: PPO with SIL improves sample efficiency in sparse reward environments while maintaining stability, opening avenues for hybrid algorithms leveraging complementary learning paradigms.

References

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

Yi-Pei Chan. Deep reinforcement learning for minigrid.

Theresa Eimer, Marius Lindauer, and Roberta Raileanu. Hyperparameters in reinforcement learning and how to tune them. In *International conference on machine learning*, pages 9104–9149. PMLR, 2023.

Linxi Fan, Guanzhi Wang, Yunfan Jiang, Ajay Mandlekar, Yuncong Yang, Haoyi Zhu, Andrew Tang, De-An Huang, Yuke Zhu, and Anima Anandkumar. Minedojo: Building open-ended embodied agents with internet-scale knowledge. *Advances in Neural Information Processing Systems*, 35: 18343–18362, 2022.

Bangzheng Li, Ningshan Ma, and Zifan Wang. Rewarded region replay (r3) for policy learning with discrete action space. *arXiv preprint arXiv:2405.16383*, 2024.

Junhyuk Oh, Yijie Guo, Satinder Singh, and Honglak Lee. Self-imitation learning. In *International conference on machine learning*, pages 3878–3887. PMLR, 2018.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

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

Acknowledgments

The code was inspired by the SLM-Lab and CleanRL framework demonstrated in the Hugging Face Deep RL course:
`https://huggingface.co/learn/deep-rl-course/unit8/hands-on-cleanrl`
`https://github.com/kengz/SLM-Lab`
`https://github.com/TianhongDai/self-imitation-learning-pytorch`

Checklist

This checklist is not mandatory, but can help you set up your project in a reproducible manner. Options for filling it out are: [Yes] [No] [NA]

1. General points:
   (a) Do the main claims made in the abstract and introduction accurately reflect your contributions and scope? [Yes]
   (b) Did you cite all relevant related work? [Yes]
   (c) Did you describe the limitations of your work? [Yes]
   (d) Did you include a discussion of future work? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes]
3. If you ran experiments (e.g. for benchmarks)...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you run at least 10 repetitions of your method? [Yes]
   (d) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (e) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] Hyper param table
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you make sure the license of the assets permits usage? [Yes]
   (c) Did you reference the assets directly within your code and repository? [Yes]