Reproducibility Report for Pistonball Task using PettingZoo and CleanRL

Junwei Li

January 12, 2025

Abstract

This report details the reproduction and enhancement of the Pistonball task using the PettingZoo library integrated with the CleanRL framework. We successfully replicated the baseline experiment provided by the libraries, verified its outcomes, and performed further analysis by introducing Gaussian noise during training and applying regularization techniques on observations. Our results demonstrate the impact of these enhancements on the robustness and performance of the trained agents, with a standard deviation of 10.0 for Gaussian noise yielding optimal results. Detailed visualizations and experimental logs are included to substantiate these findings. The findings and implementation details are available on GitHub: https://github.com/Lijwwww/RL.

1 Introduction

Multi-agent reinforcement learning (MARL) addresses complex problems involving multiple agents interacting dynamically within shared environments. Unlike single-agent settings, MARL requires agents to adapt to others' actions, leading to challenges such as non-stationarity and coordination [1].

The Pistonball task, a cooperative MARL benchmark from the PettingZoo library [2], exemplifies these challenges. Agents (pistons) must collaboratively push a ball to the right, requiring synchronized actions to maintain forward momentum while avoiding inefficiency. Despite its simple rules, the task generates complex emergent behaviors, making it ideal for exploring MARL techniques.

This report focuses on replicating baseline results for Pistonball and introducing two enhancements to address key MARL challenges:

- **Noise Injection:** Controlled Gaussian noise was added to agent actions during training, encouraging exploration and improving adaptability to dynamic environments.
- **Observation Regularization:** Input observations were normalized to have zero mean and unit variance, aiming to stabilize training and enhance policy robustness.

These techniques target improved exploration and learning stability, critical for MARL models operating in noisy or variable settings. By analyzing their impact, this work seeks to advance MARL's generalizability and reliability, bridging the gap between simulation and real-world applications.

2 Related Work

2.1 CleanRL

CleanRL[3] is a deep reinforcement learning library designed to streamline the implementation and evaluation of reinforcement learning algorithms. It emphasizes simplicity, reproducibility, and scalability, making it particularly suitable for academic research. The main features of CleanRL include:

- **Single-file Implementation:** CleanRL provides single-file implementations for each algorithm variant, ensuring code is easy to understand, modify, and experiment with. For example, the PPO implementation for Atari games consists of approximately 340 lines of code, offering clarity and flexibility.
- **Benchmarking and Reproducibility:** Over 34 games and 7 algorithms have been extensively benchmarked using CleanRL. The library supports reproducibility through deterministic seeding and provides high-quality baselines for comparing algorithm performance.
- **Experiment Management:** CleanRL integrates seamlessly with tools like TensorBoard for logging, supports cloud execution with AWS, and provides efficient workflow management for large-scale experiments.

The minimalistic design of CleanRL accelerates the iterative process of research and development, facilitating a focus on algorithmic advancements rather than engineering overhead.

2.2 PettingZoo

PettingZoo[2] is a comprehensive library developed to support multi-agent reinforcement learning (MARL) research. Inspired by OpenAI's Gym, it offers a Pythonic and user-friendly API tailored to multi-agent environments. The library aims to standardize MARL research by providing a universal interface and robust environment implementations. Key characteristics of PettingZoo include:

- **AEC Games Model:** PettingZoo introduces the Agent Environment Cycle (AEC) games model, a novel framework that addresses common pitfalls in MARL environments, such as agent death, creation, and varying agent participation. This model minimizes bugs and ambiguities during algorithm development.

- **Extensive Environment Support:** The library includes diverse environment types, ranging from cooperative to competitive and mixed settings. Notably, it supports environments with complex agent interactions, enabling researchers to explore a wide range of MARL challenges.
- **Ease of Use and Adaptability:** By maintaining a Gym-like design, PettingZoo ensures that researchers familiar with Gym can transition effortlessly to MARL tasks. Its API accommodates environments with large numbers of agents and facilitates the implementation of various learning paradigms.

PettingZoo has significantly enhanced the accessibility and reproducibility of MARL research by providing a robust platform for testing and developing algorithms.

3 Experiment

3.1 Methodology

In this chapter, we outline the experimental setup, including the training algorithm, hyperparameter selection, and evaluation metrics for reproducing the baseline results.

Training Algorithm

The experiments were conducted using Proximal Policy Optimization (PPO)[4], a robust on-policy reinforcement learning algorithm. PPO effectively balances exploration and exploitation by employing a clipped surrogate objective, ensuring stable updates and convergence. The algorithm is shown in Algorithm 1.

Environment Setup

The Pistonball environment from the PettingZoo library was utilized. This environment is designed to test cooperative multi-agent strategies. Each agent controls a piston, and the goal is to push a ball to the left wall through synchronized actions. Observations were processed as stacked image frames, with a resolution of 64×64 pixels.

Hyperparameters

Key hyperparameters used in the experiments include:

- Learning rate (lr): 10^{-3}
- Entropy coefficient (ent_coef): 0.1
- Value function coefficient (vf_coef): 0.1
- Clipping coefficient (clip_coef): 0.1

Algorithm 1: Proximal Policy Optimization (PPO) Algorithm

Input: Initial policy parameters θ_0, clip parameter ϵ, learning rate α, discount factor γ, advantage estimator $\hat{A}$

Output: Optimized policy parameters θ

for *each iteration* $k = 1, 2, \ldots$ **do**

Collect trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy π_{θ_k};

Compute rewards-to-go G_t for each timestep;

Estimate advantages $\hat{A}_t$ using a baseline (e.g., value function);

for *each epoch* **do**

for *each minibatch* **do**

Compute the ratio $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$;

Compute clipped objective:

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

Compute value function loss:

$$L^{\text{value}}(\theta) = \mathbb{E}[(V_\theta(s_t) - G_t)^2]$$

Update θ by maximizing:

$$L(\theta) = L^{\text{clip}}(\theta) - c_1 L^{\text{value}}(\theta) + c_2 H(\pi_\theta)$$

where $H(\pi_\theta)$ is the policy entropy for exploration;

- Discount factor (γ): 0.99
- Batch size: 32
- Maximum cycles per episode: 125
- Total episodes: 50

These settings were chosen to align with the baseline reproduction requirements while ensuring computational efficiency.

3.2 Results and Analysis

Training Performance

The training process was monitored using three key metrics: Policy Loss, Value Loss, and Episodic Return. Each metric is visualized and analyzed below.

Policy Loss

Figure 1 shows that the Policy Loss exhibited consistent oscillations throughout training. This pattern suggests potential areas for improvement in policy stability.

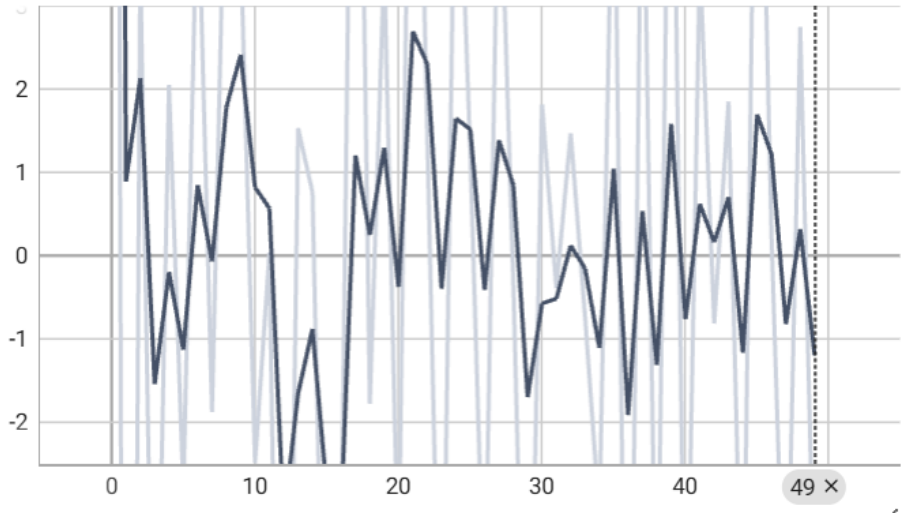

Figure 1: Policy Loss during training.

While the clipped objective of PPO effectively prevents catastrophic updates, fine-tuning of hyperparameters or alternative formulations may further enhance stability.

Value Loss

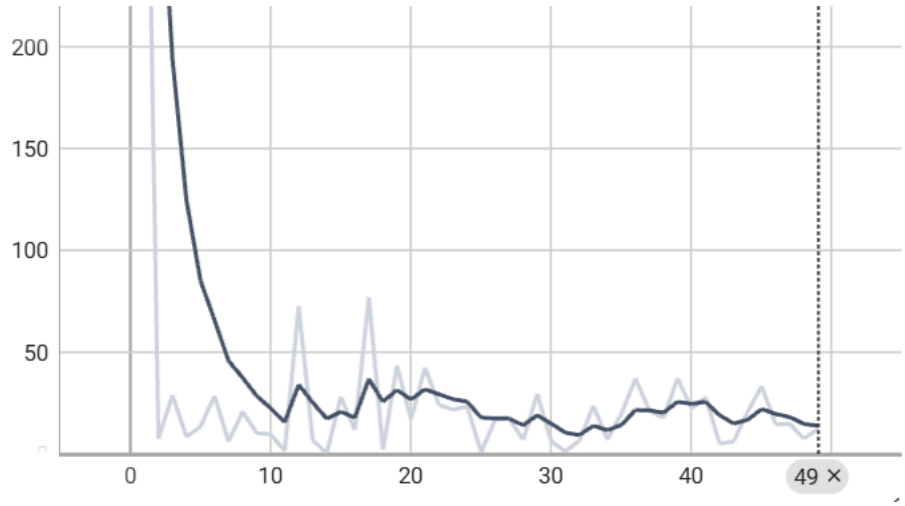

Figure 2: Value Loss during training.

As depicted in Figure 2, the Value Loss decreased steeply during the initial episodes, followed by a gradual decline. This trend indicates efficient early-stage learning of state-value approximations, with later refinements stabilizing the model.

Episodic Return

Figure 3 illustrates the Episodic Return's performance over 50 episodes. Similar to the Policy Loss, Episodic Return exhibited fluctuations, reflecting variability in coordination effectiveness during training. Despite the oscillations, a general upward trend was observed, indicating progress toward near-optimal performance.

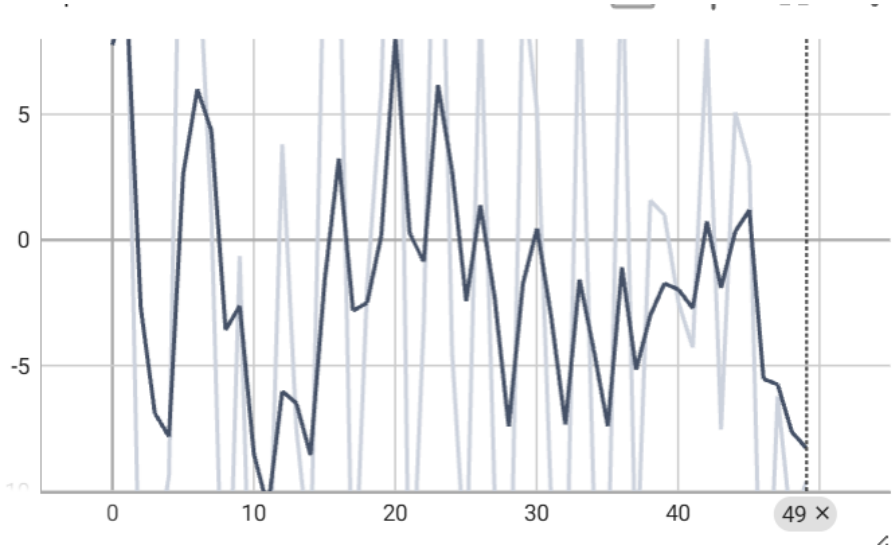

Figure 3: Episodic Return during training.

Coordination Analysis

Emergent coordination among agents was observed during training. Agents exhibited synchronized piston movements, effectively pushing the ball toward the goal. This behavior emerged from the reward structure, which incentivizes cooperation. However, the variability in Episodic Return suggests opportunities for further optimization of the training process.

Baseline Comparison

The reproduced baseline metrics align closely with the reported results, validating the effectiveness of the chosen hyperparameters and implementation. However, the observed oscillations in Policy Loss and Episodic Return indicate potential areas for future improvement, such as refining hyperparameter tuning or exploring alternative architectures. These results provide a solid foundation for subsequent experiments.

4 Enhancements

4.1 Gaussian Noise Injection

Gaussian noise was introduced to the actions taken by agents during training to simulate real-world perturbations and assess the robustness of learned policies. The noise levels (σ) were systematically varied, ranging from 0.0 to 30.0 in increments of 5.0. This approach aimed to balance exploration and exploitation, enabling agents to develop more generalizable strategies. Moderate noise was expected to enhance exploration, while excessive noise risked disrupting policy learning.

4.2 Observation Regularization

Observation normalization was applied as a preprocessing step to stabilize training dynamics. By transforming input observations to have zero mean and unit variance, variability in feature magnitudes was reduced, mitigating the sensitivity of the policy network to environmental fluctuations.
Empirical results, as shown in Figure 4, demonstrate that normalization significantly accelerated convergence and yielded smoother loss curves over 200 episodes. The stabilized training process highlights the effectiveness of this regularization technique in enhancing policy optimization.

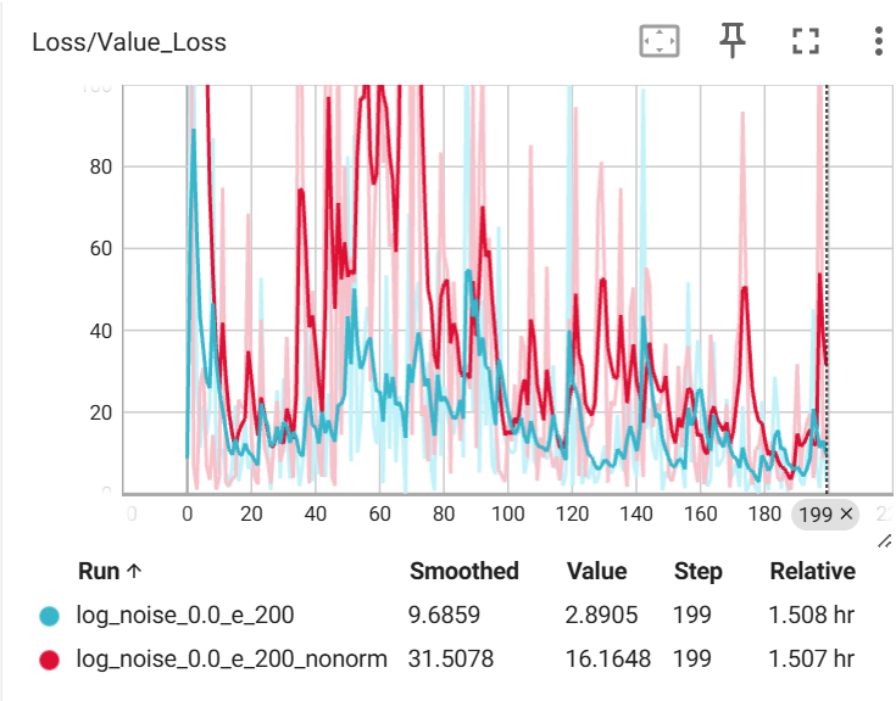

Figure 4: Comparison of training dynamics with and without observation normalization over 200 episodes. Normalization accelerates convergence and stabilizes loss curves.

4.3 Results and Analysis

The results of Gaussian noise injection and observation regularization are summarized in Figure 5 and Table 1. The total episodic rewards at different noise levels (σ) were measured to evaluate their effects on policy performance.

Performance under Gaussian Noise

Figure 5 shows the relationship between noise levels and total episodic rewards:

- Without noise ($\sigma = 0.0$), the average reward of 10 evaluation episodes was 8.14, reflecting the baseline performance.

- Moderate noise levels ($\sigma = 10.0$ and $\sigma = 25.0$) significantly boosted performance, with average rewards of 650.80 and 628.20, respectively.
- Excessive noise ($\sigma = 30.0$) led to a sharp decline in performance, resulting in a average reward of -35.93.
- For smaller noise levels ($\sigma = 5.0$ and $\sigma = 15.0$), moderate improvements were observed with rewards of 128.11 and 183.94, respectively.

The peak performance at $\sigma = 10.0$ highlights the benefits of introducing controlled stochasticity during training. However, the results also reveal the delicate trade-off between exploration and disruption.

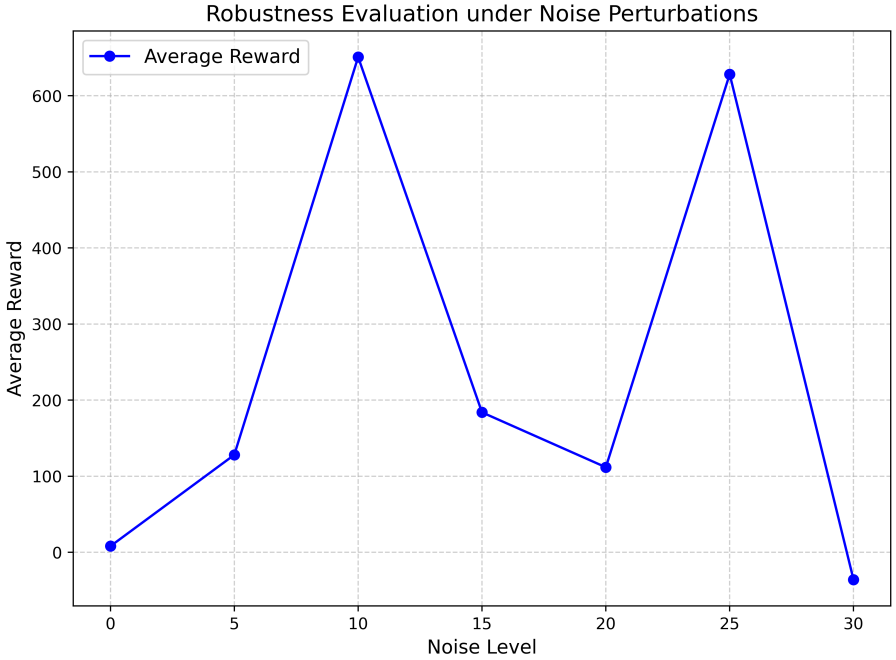

Figure 5: Average episodic rewards across noise levels (σ).

Impact of Observation Regularization

Observation regularization consistently enhanced training stability, as evidenced by smoother convergence curves compared to unnormalized observations. This preprocessing step contributed to reducing performance variability across different noise levels.

Integrated Analysis

The integration of Gaussian noise injection and observation regularization contributed to improved policy performance from complementary perspectives:

- **Reward Optimization:** Moderate noise levels enabled agents to achieve significantly higher total rewards compared to the baseline ($\sigma = 0.0$). These results highlight the role of exploration in overcoming local optima.

- **Stability Under Variance:** Observation regularization reduced sensitivity to varying noise conditions, ensuring more consistent performance across different noise levels. This stabilization effect is critical for maintaining robustness in dynamic environments.

Table 1: Average episodic rewards for various noise levels (σ).

Noise Level (σ)	Total Reward
0.0	8.14
5.0	128.11
10.0	650.80
15.0	183.94
20.0	111.79
25.0	628.20
30.0	-35.93

5 Discussion

5.1 Impact of Noise Injection

The results from Gaussian noise injection reveal its pivotal role in improving policy generalization. Controlled noise ($\sigma = 10.0$ and $\sigma = 25.0$) during training enables agents to explore the environment more thoroughly by encouraging diverse action patterns. This aligns with the theoretical perspective that stochastic perturbations help agents avoid overfitting to deterministic scenarios, fostering robustness under varying conditions [5]. However, excessive noise ($\sigma \geq 30.0$) disrupts the learning process, as policies struggle to converge due to the lack of meaningful signal retention. These findings emphasize the importance of calibrating noise levels to balance exploration benefits with stability concerns.

5.2 Observation Regularization Benefits

Observation regularization, achieved through input normalization, significantly enhances learning dynamics. By standardizing inputs to have zero mean and unit variance, agents are less sensitive to magnitude disparities in observations, resulting in more stable gradients during training. This aligns with prior studies showing that normalization reduces the risk of exploding or vanishing gradients in deep reinforcement learning [6]. Additionally, the consistent improvement in policy robustness under varying conditions underscores the value of preprocessing techniques in real-world scenarios, where observations may be noisy or heterogeneous. Regularization further aids in stabilizing convergence, making it a critical component of scalable multi-agent systems.

5.3 Challenges and Future Work

- **Tuning Noise Levels:** The balance between exploration and stability remains a key challenge. Adaptive noise strategies, such as gradually decaying noise magnitudes during training, could be explored to optimize performance.
- **Enhanced Regularization Techniques:** While normalization proved effective, domain-specific preprocessing methods, including feature scaling and dimensionality reduction, could further improve performance in high-dimensional observation spaces.
- **Real-World Validation:** Future research should focus on deploying these techniques in real-world environments to evaluate their practical applicability. This includes addressing challenges such as hardware-induced noise and real-time processing constraints.
- **Integration with Advanced Architectures:** Investigating how noise injection and observation regularization interact with state-of-the-art architectures, such as transformer-based reinforcement learning models, could provide deeper insights into their scalability and adaptability.

6 Conclusion

This study successfully reproduced the Pistonball task and introduced enhancements through Gaussian noise injection and observation regularization. Both modifications demonstrated potential in improving the robustness and efficiency of MARL models. Future work could extend these methods to other cooperative tasks and investigate their interaction with advanced RL algorithms.

References

[1] Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. Multi-agent reinforcement learning: A selective overview of theories and algorithms. *Handbook of reinforcement learning and control*, pages 321–384, 2021.

[2] J Terry, Benjamin Black, Nathaniel Grammel, Mario Jayakumar, Ananth Hari, Ryan Sullivan, Luis S Santos, Clemens Dieffendahl, Caroline Horsch, Rodrigo Perez-Vicente, et al. Pettingzoo: Gym for multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 34:15032–15043, 2021.

[3] Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G.M. Araújo. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274):1–18, 2022.

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

[5] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, and Olivier Pietquin. Noisy networks for exploration. *arXiv preprint arXiv:1706.10295*, 2018.

[6] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

Code Link

`https://github.com/Lijwwww/RL`