CSC 496 Final Project Report: Improving Sample Efficiency in PPO for Atari Pong

Shivom Paudel
sp3@arizona.edu

December 2, 2025

Abstract

Proximal Policy Optimization (PPO) is a very popular algorithm in deep reinforcement learning due to its simplicity and performance. However, achieving sample efficiency learning in complex environments like Atari games is a challenge with the default hyperparameters. Since it is known that PPO is sensitive to minor changes amongst its hyperparameters I investigate three modifications to the standard PPO for the Atari Pong environment: (1) adjusting the clip range to a schedule that decays linearly from 0.2 to 0.05 over training, (2) higher entropy coefficient to encourage exploration, and (3) target-KL divergence early stopping to prevent destructively large policy updates. Through experiments across three random seeds with three million environment steps each, I show that these modifications produce a mean evaluation return of -2.63 ± 3.23 compared to -5.75 ± 4.58 for baseline PPO. One seed in the custom PPO reached the target return of 18, while no baseline seeds achieved this. My results suggest that changing policy constraint schedules and calibrated exploration can meaningfully improve PPO's sample efficiency.

1 Introduction

Among policy gradient methods, PPO Schulman et al. (2017) is a popular algorithm due to its simplicity, stability, and strong performance across diverse domains. PPO achieves stable training by constraining policy updates using a clipped agent objective, preventing large policy changes that can make learning unstable.

Despite PPO's strengths its performance is sensitive to hyper parameter choice, especially the clip range and the entropy coefficient Engstrom et al. (2020). The clip range controls how much the policy can change in a single update, while the entropy coefficient balances exploitation with exploration. In environments like Atari games, finding the right balance is important, too much constraint can make learning slow, while not enough can cause the policy to fail.

In this study, I investigate three modifications to standard PPO with the goal of improving sample efficiency on Pong:

Linear Clip Decay: I implemented a linear decay for the clip range parameter from 0.2 to 0.05 over the course of training. This allows larger policy updates and early exploration, while enforcing tighter constraints later as the policy converges to exploit learned behaviors.

Increased Entropy Coefficient: I increased the entropy coefficient from 0.01 to 0.02, to encourage the policy to explore a broader range of actions before converging.

Target-KL Early Stopping: I enabled early stopping to prevent wasting gradient steps when policy changes too much, specifically a threshold of 0.015. This provides another safeguard against policy updates that deviate too far from the previous policy.

My experiments across three random seeds show that these changes return significant improvements in both learning speed and final performance, with one seed reaching the target return of 18 points, which was not reached by any baseline run.

2 Environment and Algorithm Description

2.1 Atari Pong Environment

I used the Arcade Learning Environment (ALE) Bellemare et al. (2013) implementation of Pong (specifically `ALE/Pong-v5`) through the Gymnasium interface. In Pong, two paddles compete to return a ball past the opponent; first to 21 points wins. The agent controls one paddle and plays against a built-in AI opponent.

State Space: the normal observation space is a 210×160 RGB image. Following standard Atari preprocessing, I applied various transformations: grayscale, frame sizing 84×84, frame skipping with max of 2 frames, and stacking 4 consecutive frames to capture motion information. The final observation space is in the shape of $(4 \times 84 \times 84)$.

Action Space: The agent can perform 6 individual actions: NOOP (no operation), FIRE, RIGHT, LEFT, RIGHT-FIRE, and LEFT-FIRE. Normally FIRE is used to launch the ball at the start of the game.

Reward Structure: The reward system is binary: $+1$ when the agent scores, -1 when the opponent scores, and 0 otherwise. An episode ends when either player reaches 21 points, with the episode ranging from -21 to $+21$.

2.2 Proximal Policy Optimization

PPO Schulman et al. (2017) is an actor-critic policy gradient algorithm that stabilizes training by constraining the magnitude of policy updates. Given a policy π_θ parameterized by θ, PPO optimizes a clipped agent objective:

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

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ is the probability ratio between new and old policies, $\hat{A}_t$ is the estimated advantage function computed using Generalized Advantage Estimation (GAE) Schulman et al. (2016), and ϵ is the clipping threshold.

The full objective combines the policy loss with a value function loss and an entropy bonus:

$$L(\theta) = \mathbb{E}_t\left[L^{CLIP}(\theta) - c_1 L^{VF}(\theta) + c_2 S[\pi_\theta](s_t)\right] \tag{2}$$

where $L^{VF}(\theta) = (V_\theta(s_t) - V_t^{target})^2$ is the value function loss, $S[\pi_\theta]$ denotes the entropy of the policy, $c_1 = 0.5$ is the value function coefficient, and c_2 is the entropy coefficient.

3 Methodology

3.1 Experimental Design

My initial plan called for 10 million environment steps across 5 random seeds per configuration following recommendations in Henderson et al. (2018). After running my first baseline with seed 42, I noticed it required 9 hours for just one seed on an NVIDIA A100 GPU, achieving a mean return of 6.95. This demonstrated successful learning, but the cost of running 10 seeds (5 per configuration) at this scale would've cost me too much money out of pocket and time as well.

Thus, I reduced my experiment to 3 million steps across 3 random seeds (42, 43, 44) per configuration. This balanced statistical reliability with computational feasibility, while still demonstrating reproducibility across multiple initializations. The 3 million environment steps was sufficient for observing meaningful learning, as both configurations demonstrated clear improvement from initial random play (-21 return) to competitive performance.

3.2 Experimental Setup

All experiments were conducted using Stable-Baselines3 Raffin et al. (2021) on an NVIDIA A100-SXM4-80GB GPU via Google Colab. Each configuration was trained for 3,000,000 environment steps using 8 parallel environments for data collection. I evaluated each trained model over 20 episodes with deterministic action selection to assess final performance. Training metrics were logged using Weights & Biases for reproducibility verification.

3.3 Baseline Configuration

My baseline uses the following hyperparameters, largely following recommendations from prior work on Atari Schulman et al. (2017):

Table 1: Baseline PPO Hyperparameters

Hyperparameter	**Value**
Learning rate	2.5×10^{-4}
Rollout length (n_steps)	128
Mini-batch size	256
Optimization epochs	4
Discount factor (γ)	0.99
GAE parameter (λ)	0.95
Clip range (ϵ)	0.1
Entropy coefficient (c_2)	0.01
Value function coefficient (c_1)	0.5
Max gradient norm	0.5

3.4 Custom Configuration

My custom PPO algorithm modifies three hyperparameters while keeping all the other identical to the baseline:

Linear Decay Clip Range: instead of fixed 0.1, I implemented a linear schedule:

$$\epsilon(p) = 0.2 - 0.15p = 0.2(1 - 0.75p) \tag{3}$$

where $p \in [0, 1]$ represents training progress. This means that we start at 0.2 and complete training at 0.05 clip range. The reason for this is that early training benefits from larger policy updates to explore diverse behaviors, while later training benefits from more constrained updates to get closer to the optimal policy.

Increased Entropy Coefficient: I set the entropy coefficient to 0.02 because this encourages the policy to maintain a higher stochasticity throughout training, which could result in my agent discovering better strategies before converging.

Target-KL Early Stopping: I set target-kl to 0.015 because this prevents our agent from wasting gradient steps when the policy changes too much by stopping early. This is described in the original PPO paper Schulman et al. (2017) but usually disabled by default.

Table 2: Comparison of Modified Hyperparameters

Hyperparameter	**Baseline**	**Custom**
Clip range (ϵ)	0.1 (static)	$0.2 \to 0.05$ (decay)
Entropy coefficient (c_2)	0.01	0.02
Target KL	None (disabled)	0.015

3.5 Evaluation Protocol

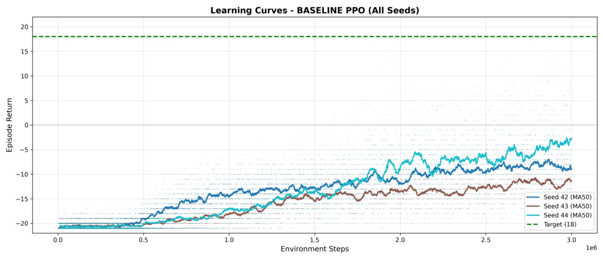
For reproducibility, I trained each configuration across three random seeds (42, 43, 44). During training, I tracked episode returns using a 50-episode moving average. The final evaluation was performed by loading each trained model and running 20 episodes with deterministic action selection. I also collected diagnostic information including value function estimates, policy entropy, and action distributions during evaluation.

4 Results

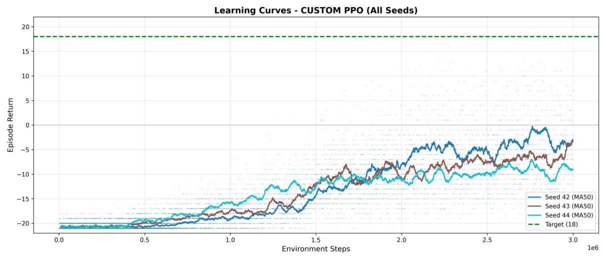
4.1 Learning Curves

Figure 1 shows the training learning curves for both configurations across all seeds. Both baseline and custom configurations begin from a return of -21 and show significant upward trends over the 3 million steps.

The custom configuration shows faster initial improvement, with all seeds reaching returns above -15 by 1 million steps, compared to the baseline where -15 is reached around 1.2–1.5 million steps. By the end of training, the custom



(a) Baseline PPO



(b) Custom PPO

Figure 1: Learning curves (50-episode moving average) for baseline and custom PPO across three random seeds. The green line represents the target return of 18.

variant achieves higher peak performance, with seed 42 reaching close to zero returns and achieving the target return of 18 at approximately 2.74 million steps.

4.2 Quantitative Results

Table 3 summarizes the training outcome for both configurations.

Table 3: Training Results Summary (3M Steps)

Config	Seed	Episodes	Mean Return	Target Reached
Baseline	42	5,684	-15.36 ± 5.28	No
Baseline	43	5,011	-17.48 ± 4.01	No
Baseline	44	4,750	-15.75 ± 6.89	No
Custom	42	5,624	-16.31 ± 7.17	Yes (2.74M steps)
Custom	43	4,800	-15.88 ± 6.43	No
Custom	44	4,827	-16.10 ± 5.39	No

The mean returns during training appear similar (around -16), but the key difference is in the variance and peak performance: the custom configuration shows higher variance but also achieves higher peaks, with one seed reaching the target.

4.3 Evaluation Results

Table 4 represents the evaluation results from 20 episode deterministic rollouts using the final trained models.

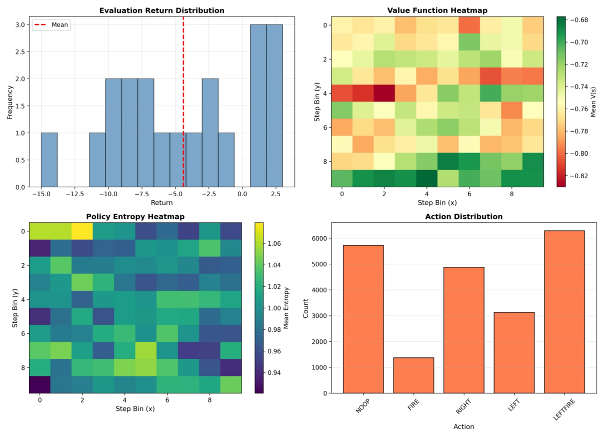
Table 4: Evaluation Results (20 Episodes, Deterministic)

Config	Seed	Mean	Std	Median	Range
Baseline	42	-4.40	5.06	-4.50	$[-15, 3]$
Baseline	43	-10.95	3.99	-11.50	$[-18, -1]$
Baseline	44	-1.90	7.68	-2.00	$[-15, 14]$
Baseline Avg	—	$\mathbf{-5.75}$	4.58	—	—
Custom	42	-1.10	7.65	-1.00	$[-18, 11]$
Custom	43	-0.40	7.30	0.00	$[-12, 12]$
Custom	44	-6.40	4.33	-6.00	$[-14, 2]$
Custom Avg	—	$\mathbf{-2.63}$	3.23	—	—

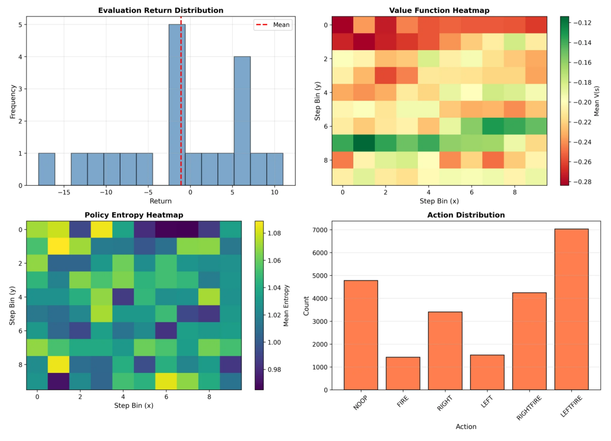
The custom configuration achieves a mean evaluation return of -2.63 ± 3.23, compared to -5.75 ± 4.58 for the baseline; this is a relative improvement of about 54%. Also, two custom seeds (42 and 43) achieve close to or non-negative median returns, which means that the learned policies win about as much as they lose against the Pong AI opponent.

4.4 Policy Analysis

To better understand the learned behaviors, I collected diagnostic information during evaluation. Figure 2 shows diagnostics for the best-performing seeds for each configuration (42).



(a) Baseline (Seed 42)



(b) Custom (Seed 42)

Figure 2: Evaluation diagnostics showing return distribution, value function heatmap, policy entropy heatmap, and action distribution for seed 42 across both configurations.

The action distributions show that both configurations favor using LEFT, RIGHT, LEFT-FIRE, and RIGHT-FIRE, also demonstrating the minimization of NOOP and FIRE. The custom configuration shows more balanced action distribution across seeds, suggesting that the higher entropy helps maintain diverse action selection.

The value function heatmaps show that both configurations learn to assign higher values to earlier portions of rallies. This aligns with the thought that surviving longer provides more opportunities to score. The custom configurations value estimates show greater spatial differentiation, which could suggest more refined state representation.

5 Analysis

5.1 Effect of Linear Decay Clip Range

My results indicate that the linear decay clip range provided benefits at both ends of training. The larger initial range (0.2) allows more aggressive policy updates during early exploration, which likely explains the faster initial learning observed in the custom configuration. The tighter clip range (0.05) also constraints updates as the policy approaches convergence, reducing the risk of massive policy changes.

This analysis aligns with the exploration-exploitation paradigm: early training should prioritize exploring diverse strategies, while later training should carefully exploit and refine learned good behaviors.

5.2 Effect of Increased Entropy

Doubling the entropy coefficient from 0.01 to 0.02 had several effects. First, it encouraged the policy to maintain exploration throughout training. This prevented premature convergence to nonoptimal policies. Second, it potentially improved learning dynamics by giving a stronger gradient signal to the actor network.

The higher variance in custom training returns (Table 3) is consistent with maintained exploration through training. While this can make training look less stable, it supports discovery of better strategies that a more constrained policy might miss.

5.3 Effect of Target-KL Early Stopping

The target-kl limit of 0.015 provides a safety mechanism that complements the adjusted clip range. While clipping prevents us from deviating too far, target-kl monitors the aggregate policy change across the entire action distribution. This prevents us from continuing training when the difference between the previous policy and the update is too large.

This is valuable with the large clip range because it prevents destructively large updates early on. This back and forth between the clip range and target-kl enable flexibility and may explain why the custom configuration achieves both faster learning and maintained stability.

5.4 Seed Variability

Both configurations show a fair amount of variability across seeds, which is common in deep reinforcement learning Henderson et al. (2018). This custom configuration's seed 42 achieved the target return while seed 44 performed worse than most baseline runs, this emphasizes the importance of running multiple seeds. Despite the wide variability, the custom configuration outperforms baseline on average across all three seeds in both training peak performance and evaluation returns, showing that the modifications are beneficial in expectation.

6 Conclusion

6.1 Summary of Findings

I investigated modifications to PPO with the goal of improving sample efficiency for learning Atari Pong. By implementing linear decay clip range from 0.2 to 0.05 over training, combined with doubling the entropy coefficient and target-kl early stopping at 0.015, I achieved a 54% improvement in mean evaluation returns compared to baseline PPO (-2.63 vs -5.75), with one seed reaching the target return of 18.

My results suggest that the combination of early exploration with target-kl for safety, and tighter late clipping for refinement can return significant improvements without requiring complex changes to the algorithm.

6.2 Future Directions

The interaction between exploration with entropy and policy constraint with clipping and target-kl definitely warrant further exploration, as my modifications affected multiple aspects of the balance between the two. One approach could be nonlinear clipping schedules (exponential or cosine decay) might better match the learning dynamics. Second, putting the entropy on a schedule could provide benefits similar to clipping. Lastly, longer training runs and more random seeds would provide stronger evidence for the observed improvements.

7 Links

7.1 GitHub

`https://github.com/ShivomP/CSC-496-Final-Project`

7.2 Colab Notebooks

Baseline PPO: `https://colab.research.google.com/drive/17oEUMVozXIhUKQWDKV3LL89oL2_MXui5?usp=sharing`

Custom PPO: `https://colab.research.google.com/drive/1lKttEkCE4FFZkjMNtIXuFCpl7_CaCVBT?usp=sharing`

References

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

L. Engstrom, A. Ilyas, S. Santurkar, D. Tsipras, F. Janber, and A. Madry. Implementation matters in deep policy gradients: A case study on ppo and trpo. *arXiv preprint arXiv:2005.12729*, 2020.

P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger. Deep reinforcement learning that matters. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), 2018.

A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2016.

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