Lunar Lander Continuous Control with Proximal Policy Optimization and Generalized Advantage Estimation

Ken K. Hong
khong18@gatech.edu
Georgia Institute of Technology

Abstract

This project studies reinforcement learning applied to continuous control tasks in the `LunarLanderContinuous-v3` environment. Continuous control differs from discrete action spaces by involving an infinite set of possible actions. Proximal Policy Optimization combined with Generalized Advantage Estimation improves learning stability and performance. More broadly, applying PPO to large language models like ChatGPT on human feedback sharpens their alignment with user intent, resulting in more accurate and reliable outputs. Experimental results highlight the impact of hyperparameters such as `GAE` λ, `epsilon_decay`, and `hidden_size`, along with the trade-off between model complexity and efficiency. Future work will explore alternative reinforcement learning algorithms, enhance data efficiency, and strengthen generalization. Challenges in real-world deployment include performance variability and environmental complexity. These findings offer insights into the strengths and limitations of reinforcement learning in complex continuous control tasks.

1 Introduction

This project explores applying reinforcement learning to complex sequential decision-making problems within continuous control environments. Unlike traditional methods such as tabular SARSA or Q-learning, which operate with a fixed set of discrete actions, continuous control tasks require agents to select from an unlimited range of actions. (Sutton & Barto, 2018) To illustrate the approach, the `LunarLanderContinuous-v3` environment from the Gymnasium library is used, where the agent learns to control the lunar lander's engines to ensure a smooth and accurate landing.

Proximal Policy Optimization (PPO) combined with Generalized Advantage Estimation (GAE) is chosen as the reinforcement learning agent for this task. PPO is selected for its strong performance, training stability, and relative simplicity compared to more complex algorithms like Trust Region Policy Optimization (TRPO), as well as its advantages over other popular methods such as TD3 and DDPG (Schulman et al., 2017). PPO has also been used effectively in the alignment and fine-tuning of large language models, including ChatGPT (Ouyang et al., 2022; Unsloth, 2025). This report outlines the problem setup, rationale for the chosen approach, actor-critic architecture, training process, and agent evaluation, concluding with key insights and suggestions for future work.

2 Lunar Lander Continuous Control Environment

The goal is to train a reinforcement learning agent to solve the `LunarLanderContinuous-v3` environment from the Gymnasium library. This environment presents a control challenge based on a physics simulation of a lunar lander. The agent's objective is to learn a policy that guides the lander from a random initial position near the top of the screen to a smooth and accurate landing at the designated target located at coordinates $(0, 0)$. The state space, action space, and reward function are defined as described in the Gymnasium documentation (Foundation, 2025).

2.1 State Space

At each timestep, the agent observes an 8-dimensional state vector that captures the lander's physical status. The state space S is defined as:

$$s_t = (x, y, \dot{x}, \dot{y}, \theta, \dot{\theta}, \text{leg}_L, \text{leg}_R)$$

where:

- x, y: The horizontal and vertical position coordinates of the lander.
- $\dot{x}, \dot{y}$: The horizontal and vertical velocities.
- $\theta, \dot{\theta}$: The angle and angular velocity of the lander.
- $\text{leg}_L, \text{leg}_R$: Binary indicators showing whether the landing legs are touching the ground.

2.2 Action Space

The agent interacts with the environment through a two-dimensional continuous action space A, which controls the lander's main and side engines. The action vector is defined as:

$$a_t = (a_{\text{main}}, a_{\text{side}})$$

where:

- **Main Engine Throttle** a_{main}: a value in the range $[-1, 1]$ with a non-linear response:
  - If $a_{\text{main}} \in [-1, 0]$, the main engine is off.
  - If $a_{\text{main}} \in (0, 1]$, the throttle is smoothly adjusted from 50% to 100% power.
- **Side Engine Throttle** a_{side}: a value in the range $[-1, 1]$ controlling the left and right orientation thrusters:
  - If $a_{\text{side}} \in [-0.5, 0.5]$, neither side engine fires.
  - If $a_{\text{side}} \in (0.5, 1]$, the right engine fires, scaling smoothly from 50% to 100% power.
  - If $a_{\text{side}} \in [-1, -0.5)$, the left engine fires, scaling smoothly from 50% to 100% power.

2.3 Rewards and Termination Condition

The reward structure guides the agent toward a successful landing by combining several components:

- **Terminal Rewards** provide a positive reward of $+100$ when the lander comes to a complete stop on the landing pad. Conversely, a penalty of -100 is applied if the lander crashes.
- **Shaping Rewards** offer continuous feedback by rewarding the agent for moving closer to the landing pad and penalizing it for moving away.
- **Auxiliary Rewards**: $+10$ points for each landing leg that touches the ground.
- **Penalties**: -0.3 points for each thruster use to encourage fuel efficiency.

The reward structure requires balancing immediate minor penalties, such as fuel consumption, against larger but delayed rewards, including a successful landing. The discount factor γ is set to 0.99 for this episodic task, promoting the development of a policy that prioritizes the significant future reward of a safe landing over the small immediate cost of engine use.

An episode terminates when the lander crashes, lands successfully, or moves beyond the visible area. The problem is considered solved when the agent achieves an average score of 200 or higher over 100 consecutive evaluation episodes.

2.4 Environmental Factors

The environment simulates physical forces that influence the lander's behavior:

- **Gravity** pulls the lander downward, typically set between 0 and -12.
- **Wind** introduces random variation to the lander's movement when enabled.
- **Wind Power** controls the strength of linear wind forces.
- **Turbulence Power** determines the intensity of rotational wind effects.

All values in this project use the default settings.

3 Clipped Proximal Policy Optimization Implementation with GAE

Clipped Proximal Policy Optimization (PPO) combined with Generalized Advantage Estimation (GAE) provides a robust and effective solution for this task. PPO was chosen over A2C, TD3, and DDPG due to superior training stability, simpler implementation, and success on continuous control benchmarks (Schulman et al., 2017). The clipped objective with GAE mitigates large policy updates that destabilize training, resulting in more reliable performance and simplified tuning. Detailed information on PPO implementation is available in (OpenAI, 2017).

3.1 The Actor-Critic Architecture

PPO uses an actor-critic framework consisting of two neural networks trained simultaneously:

- **Actor (Policy Network, π_θ)**: Defines the agent's behavior by mapping the current state s_t to a stochastic policy. The policy network is implemented as a four-layer multilayer perceptron (MLP) with dropout and ReLU activation functions, constructed using PyTorch. The output layer employs a tanh activation to keep actions within the desired range.
- **Critic (Value Network, V_ϕ)**: Evaluates the quality of actions by estimating the expected discounted future reward from state s_t. This network is also a four-layer MLP with dropout and ReLU activations, implemented in PyTorch, with a tanh activation in the output layer.

3.2 The PPO Clipped Surrogate Objective

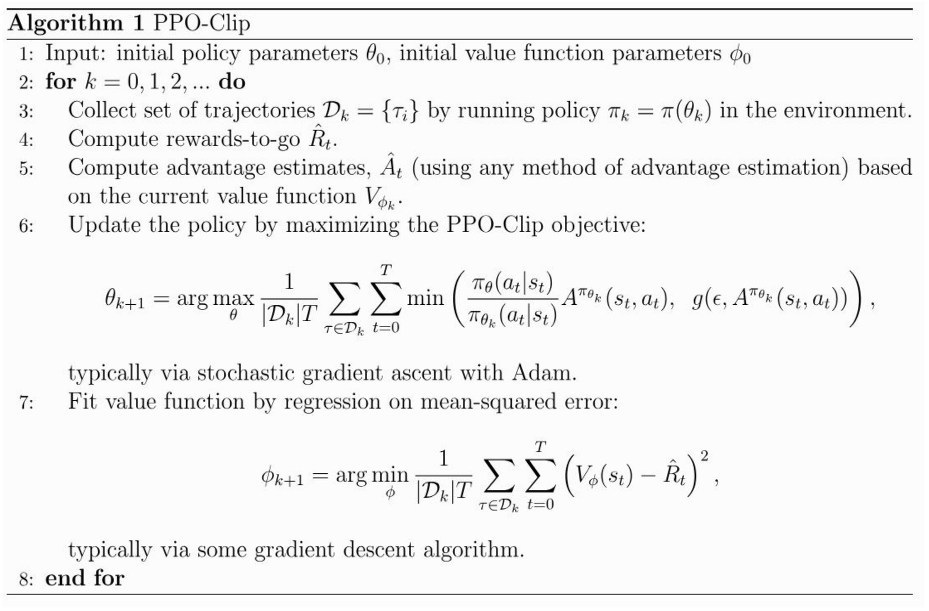
PPO's key innovation is the clipped surrogate objective, which uses the probability ratio between new and old policies to limit updates, promoting stable and efficient learning.

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}$$

The objective function is defined as:

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

In this expression, $\hat{A}_t$ denotes the estimated advantage, and ϵ specifies the clipping threshold. This approach restricts excessively large updates and avoids the computational complexity associated with methods such as TRPO. Figure 1 presents the pseudocode for PPO implementation.



Algorithm 1 PPO-Clip

```
1: Input: initial policy parameters $\theta_0$, initial value function parameters $\phi_0$
2: for $k = 0, 1, 2, \ldots$ do
3:     Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
4:     Compute rewards-to-go $\hat{R}_t$.
5:     Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based
       on the current value function $V_{\phi_k}$.
6:     Update the policy by maximizing the PPO-Clip objective:
           $\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left( \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t), \quad g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$
       typically via stochastic gradient ascent with Adam.
7:     Fit value function by regression on mean-squared error:
           $\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left( V_\phi(s_t) - \hat{R}_t \right)^2,$
       typically via some gradient descent algorithm.
8: end for
```

Figure 1: Pseudocode for the PPO algorithm. (OpenAI, 2017)

3.3 Generalized Advantage Estimation (GAE)

The effectiveness of policy updates depends on the accuracy of the advantage estimate, defined as $\hat{A}_t = Q(s_t, a_t) - V(s_t)$. GAE enhances this by balancing bias and variance (Schulman et al., 2015).

The single-step temporal difference (TD) error,

$$\delta_t = r_t + \gamma V(s_{t+1}) - V(s_t),$$

has low variance but may introduce bias due to reliance on the critic's prediction. In contrast, Monte Carlo returns offer an unbiased estimate by using full episode rewards but often suffer from high variance caused by noisy long-term outcomes.

GAE combines these methods through an exponentially weighted sum of TD errors:

$$\hat{A}_t^{\mathrm{GAE}}(\gamma, \lambda) = (1 - \lambda) \left(\delta_t^V + \lambda(\delta_{t+1}^V) + \lambda^2(\delta_{t+2}^V) + \cdots \right) = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}$$

The parameter λ adjusts the bias-variance trade-off. Values near 0.95 have proven effective in practice (Schulman et al., 2015). Together, GAE provides a smoother and more informative signal for policy updates, while PPO controls the update size, resulting in more stable and efficient learning.

3.4 RolloutBuffer

The `RolloutBuffer` stores on-policy experience tuples, including states, actions, action log probabilities, rewards, done flags, and value estimates collected during interaction with the environment. It enables efficient batch updates by accumulating data over a rollout and clearing it after each policy update. This process allows multiple epochs of minibatch training on the same data, improving sample efficiency and stabilizing policy optimization (Brunskill et al., 2025; OpenAI, 2017) .

3.5 Training Configuration and Gradient Stabilization

Training the PPO agent requires careful optimization and gradient control to maintain stability and efficiency. This section outlines the optimization setup, including the optimizer choice, parameter initialization, and gradient clipping, based on the approach in (Engstrom et al., 2020).

Optimizer Adam optimizer was employed to update both actor and critic networks. Learning rates were determined through empirical tuning and remained constant, resulting in stable and consistent training. Future work may explore cosine decay to gradually reduce the learning rate.

Gradient Clipping To prevent unstable updates and exploding gradients, gradients were clipped to keep the global norm below 0.5, ensuring stable training and preventing divergence.

Orthogonal Initialization Weights were initialized orthogonally to preserve gradient scales during backpropagation, promoting better convergence and more stable training.

4 Experimental Setup

Baseline PPO model configured using default parameters drawn from existing literature and practical experience, establishing a consistent benchmark. Trial variants altered one or more parameters to evaluate hyperparameter effects, isolating individual impacts and examining interactions.

4.1 Hyperparameter Overview

Performance of the PPO model depends on several hyperparameters affecting training stability, exploration, and model capacity. A baseline configuration used standard values derived from practical experience. Each trial variant modified a single hyperparameter to assess the impact. Comparison metrics included learning speed, reward outcomes, and variability. Primary hyperparameters included GAE lambda balancing bias and variance, epsilon decay controlling exploration, clipping threshold enforcing update stability, and hidden layer size determining model complexity.

4.2 Tuning Process and Metrics

Experimental runs compared each variant to the baseline while tracking key metrics: episode count, average reward over the last 100 episodes, standard deviation over the last 100 episodes, and completion time in minutes and seconds to evaluate configuration impact. Table 1 details baseline values and trial variants with one hyperparameter changed in each trial.

Hyperparameter	Symbol	Description	Baseline	Trials
lr_actor	α_{actor}	Learning rate for the actor network	0.0002	0.0002
lr_critic	α_{critic}	Learning rate for the critic network	0.001	0.001
gamma	γ	Discount factor	0.99	0.99
K_epochs	K	Number of epochs per update	20	20
dropout_prob	p_{drop}	Dropout probability	0.1	0.1
num_episodes	N_{episodes}	Total number of training episodes	2500	2500
max_timesteps	$T_{\max}$	Maximum timesteps per episode	2000	2000
update_timestep	T_{update}	Timesteps before each policy update	512	512
gae_lambda	λ	GAE lambda parameter	0.95	**0.8** (trial 1)
epsilon_decay	δ_ϵ	Decay rate for exploration	0.995	**0.6** (trial 2)
eps_clip	ϵ	Clipping parameter for update size	0.2	**0.4** (trial 3)
hidden_dim	h	Number of units in hidden layers	256	**128** (trial 4)

Table 1: Summary of hyperparameters used in the PPO training process

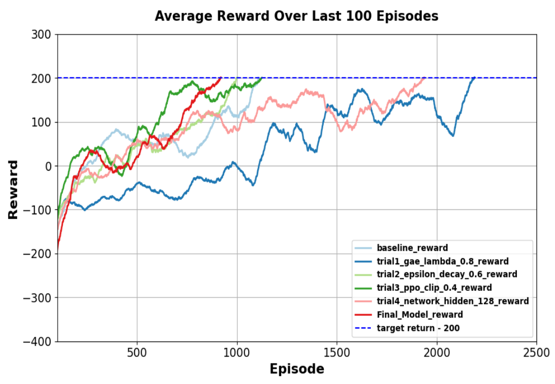
5 Experimental Results and Analysis

Training curves displayed in Figures 4–9 illustrate the effects of hyperparameters on PPO learning behavior. Reward progression and standard deviation provide empirical performance measures aligned with theoretical expectations in the Lunar Lander environment. Baseline results in Figure 4 establish a reference for evaluating selected hyperparameters using a fixed random seed of 1. Random variations may affect outcomes. Without a fixed seed, averaging multiple training runs yields a reliable assessment of hyperparameter performance.

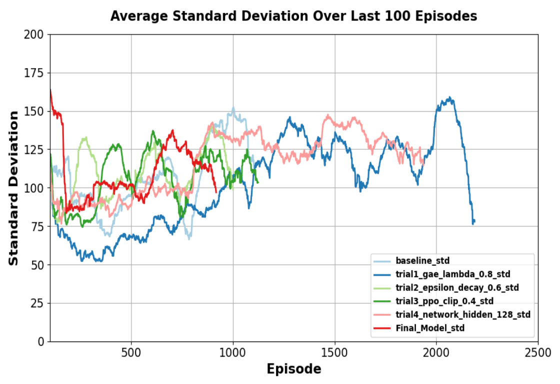
Figure 2 and Table 2 summarize baseline and variant model performance on the Colab v6e-1 TPU.

Experiment	Episodes Run	Training Time (s)	Time per Episode (s)	Average Standard Deviation
Baseline	1107	377	0.34	106.56
Trial 1 (GAE Lambda)	2189	785	0.36	101.93
Trial 2 (Epsilon Decay)	1002	449	0.45	108.75
Trial 3 (Epsilon Clip)	1125	462	0.41	105.94
Trial 4 (Hidden Dim)	1937	481	0.25	116.10
Final Model	919	361	0.39	109.96

Table 2: Experimental results



(a) Average reward across trials



(b) Standard deviation of reward across trials

Figure 2: Model performance and reward variability across trials

5.1 Impact of Hyperparameters

Effect of `gae_lambda` (Trial 1) Figure 5 illustrates the impact of the `gae_lambda` parameter on reward trajectory variance and smoothness. Lower values reduce variance but introduce bias and slow convergence. Higher values decrease bias and accelerate learning, sometimes at the expense of stability. This tradeoff aligns with theoretical distinctions between temporal-difference and Monte Carlo methods. Lowering `gae_lambda` from 0.95 to 0.8 extended training to 2189 episodes versus 1107 at baseline, indicating slower convergence. The standard deviation decreased from 106.56 to 101.93, indicating a more stable learning process.

Effect of `epsilon_decay` (Trial 2) Figure 6 illustrates the effect of `epsilon_decay` on exploration-exploitation balance during training. Faster decay rates prompt early exploitation of the learned policy, occasionally resulting in suboptimal convergence. Slower decay rates encourage extended exploration, avoiding local optima at the expense of longer training. Lowering `epsilon_decay` from 0.995 to 0.6 reduced episodes from 1107 to 1002, demonstrating faster convergence and reduced unnecessary exploration time.

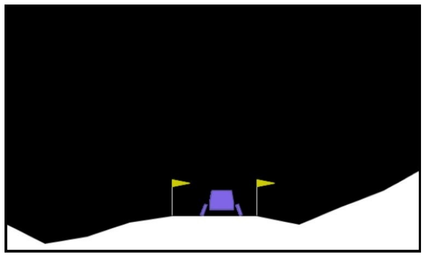


Figure 3: Lunar Lander Touchdown

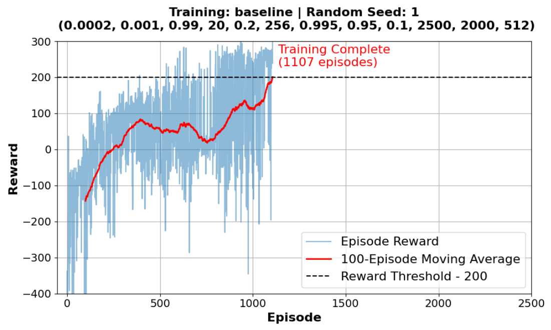


Figure 4: Baseline performance

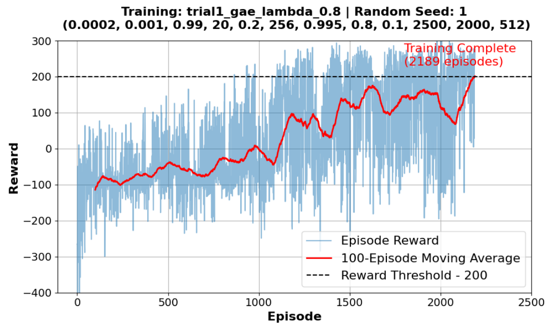


Figure 5: Trial 1 with `gae_lambda` = 0.8

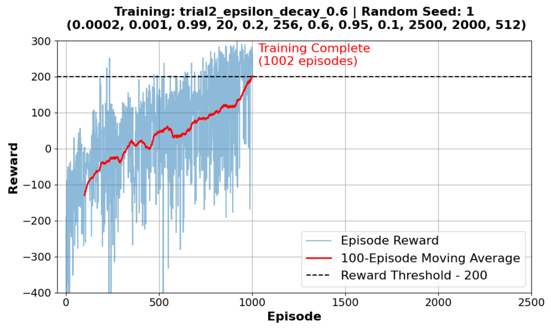


Figure 6: Trial 2 with `epsilon_decay` = 0.6

Effect of `eps_clip` (Trial 3) Figure 7 illustrates the effect of the `eps_clip` parameter on policy update magnitude. Smaller values restrict updates, promoting stability but slowing learning. Larger values allow broader updates, sometimes accelerating learning at the cost of increased instability. Trial 3 used `eps_clip`=0.4 versus 0.2 in the baseline. Similar performance and variance in both cases likely resulted from gradient clipping that limited the global gradient norm to 0.5.

Effect of `hidden_dim` (Trial 4) Figure 8 illustrates how hidden layer size influences model capacity and representational power. Larger dimensions improve performance in complex scenarios but require more computation and risk overfitting. Smaller dimensions enable faster training with reduced expressiveness. Trial 4 employed 128 hidden units instead of 256 in the baseline. Average episode time dropped from 0.34 to 0.25 seconds, while episode count rose from 1107 to 1937, reflecting increased bias from lower model complexity.

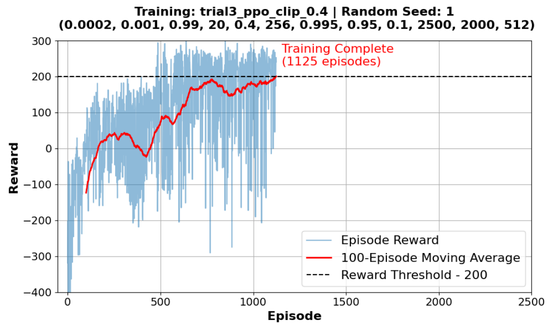


Figure 7: Trial 3 with `eps_clip` = 0.4

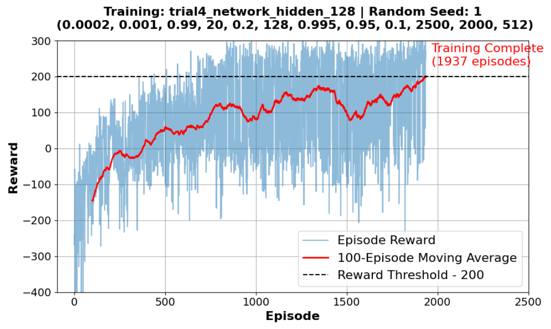


Figure 8: Trial 4 with `hidden dimension` = 128

Sensitivity to Additional Hyperparameters Learning rate, discount factor and batch size values were chosen from literature and experience. Effects were less pronounced than those of primary parameters, yet certain combinations influenced training efficiency and performance. For example, lower learning rates increased convergence time.

5.2 Final Model and Evaluation

The final model configuration was selected based on experimental results to balance convergence speed, training stability, and performance. The hyperparameters were `gae_lambda` = 0.94, `epsilon_decay` = 0.8, `eps_clip` = 0.25, and hidden layer size of 256.

This configuration resulted in faster convergence and strong returns, with only a minimal increase in the average standard deviation.Figure 9 presents the overall performance metrics, and Figure 10 shows the reward progression over the last 100 episodes.

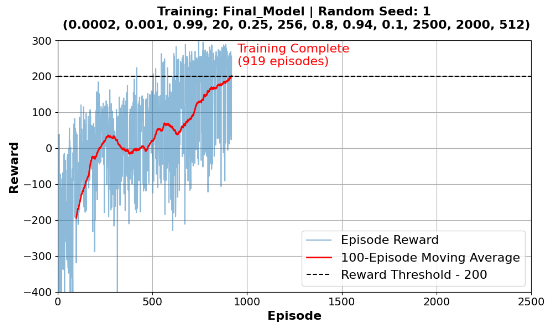


Figure 9: Final model performance

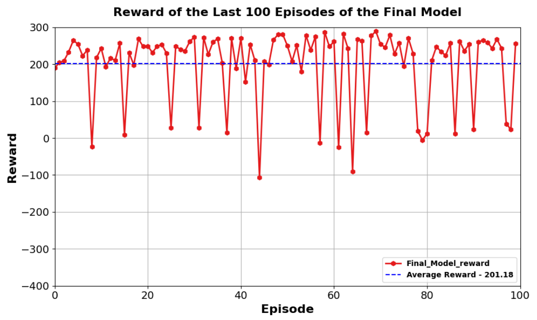


Figure 10: Reward of the last 100 episodes

6 Discussion and Conclusion

6.1 Main Takeaways

Effect of GAE Lambda Adjusting the `gae_lambda` parameter revealed a clear trade-off between bias and variance. Lower values reduced reward variance but slowed convergence, whereas higher values accelerated learning at the cost of occasional instability. This finding is consistent with existing theoretical perspectives on temporal-difference versus Monte Carlo methods.

Exploration and Stability Modifications to `epsilon_decay` and `eps_clip` influenced the exploration-exploitation balance and stability of policy updates. Faster epsilon decay promoted quicker convergence but risked suboptimal policies. Larger clipping thresholds enabled faster updates, but their effect was tied to gradient clipping.

Model Capacity and Efficiency Reducing the hidden layer size improved training speed per episode but introduced higher bias, requiring more episodes to achieve similar performance. This emphasizes the trade-off between model complexity and training efficiency.

6.2 Future Improvements

Exploring Alternative Models Future work may explore alternative reinforcement learning agents like A2C or DDPG, as Proximal Policy Optimization with GAE may be too complex for this task. Investigating function approximations such as linear or spline regression could improve training efficiency and reduce computational costs, though possibly at the expense of representation power. These methods could study learning efficiency, stability, and resource use, offering more efficient solutions.

Improved Reliability and Generalizability Running experiments with multiple random seeds and averaging results helps reduce randomness and yields a more stable evaluation of model performance. Results obtained using a fixed random seed may not fully represent the range of possible outcomes, potentially leading to biased or overly optimistic estimates. Conducting multiple trials with varied seeds provides a more reliable and generalizable measure of true model performance.

Data Efficiency Improving data efficiency through enhanced rollout buffer design and fixed training parameters during updates could be investigated in future work. Current experiments require many episodes, resulting in resource-intensive training. Techniques such as experience replay and improved sample reuse may reduce the interactions needed to achieve high performance, accelerating training and increasing practicality in real-world settings with limited or costly data.

Enhanced Agents with Direct Preference Optimization To improve performance beyond standard PPO, future research could investigate Direct Preference Optimization (DPO) frameworks. DPO, as proposed by Rafailov et al. (2023), is an alternative to reinforcement learning with human feedback (RLHF) that directly fine-tunes policies from preference data using a classification-based objective. Although originally developed for language models, the core idea of optimizing policies based on relative preferences could be adapted to the lunar landing domain. For instance, human or expert preferences over landing trajectories and safety margins could guide policy learning. Leveraging preference-based optimization may improve sample efficiency, increase robustness to rare failure modes, and lead to safer and more reliable lunar landing agents.

6.3 Scaling to Real-World Applications

Although the environment is considered "solved" with average returns exceeding 200 over 100 consecutive episodes, the final model achieved returns above 200 in only 75 percent of episodes. The remaining 25 percent showed fluctuations, including 17 episodes with near-zero rewards. These inconsistencies highlight challenges in robustness and reliability, which are critical for real-world applications where failures can result in safety risks or inefficiencies.

Variability can arise from environmental randomness, limited training data, and insufficient generalization, while real-world applications present unpredictable behavior, sensor noise, and evolving conditions. Overcoming these challenges demands domain randomization, ensemble methods, safety constraints during training, and thorough validation through realistic simulations and staged real-world trials. Although current results demonstrate promising control in simulation, further development is essential to ensure safe and reliable performance in complex real-world scenarios.

References

Emma Brunskill, Chethan Bhateja, Aishwarya Mandyam, HyunJi (Alex) Nam, Hengyuan Hu, Lansong (Ryan) Li, Shiyu Zhao, and Keenon Werling. Cs234: Reinforcement learning, 2025. URL https://web.stanford.edu/class/cs234/. Accessed: 2025-06-18.

Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. Implementation matters in deep policy gradients: A case study on ppo and trpo. *arXiv preprint arXiv:2005.12729*, 2020. URL https://arxiv.org/abs/2005.12729.

Farama Foundation. Lunar lander - gymnasium documentation, 2025. URL https://gymnasium.farama.org/environments/box2d/lunar_lander/. Accessed: 2025-06-18.

OpenAI. Proximal policy optimization, 2017. URL https://spinningup.openai.com/en/latest/algorithms/ppo.html. Accessed: 2025-06-18.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback, 2022. URL https://arxiv.org/abs/2203.02155.

Rafael Rafailov, Aditi Sharma, Eric Mitchell, Stefano Ermon, Christopher D Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *arXiv preprint arXiv:2305.18290*, 2023. URL https://arxiv.org/abs/2305.18290.

John Schulman, Philipp Moritz, Sergey Levine, Michael I Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015. URL https://arxiv.org/abs/1506.02438.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. URL https://arxiv.org/abs/1707.06347.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 2nd edition, 2018. URL http://incompleteideas.net/book/the-book-2nd.html.

Unsloth. Reinforcement learning guide, 2025. URL https://docs.unsloth.ai/basics/reinforcement-learning-guide. Accessed: 2025-06-18.