Pac-Man Gameplay with Proximal Policy Optimization and Generalized Advantage Estimation

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

Abstract

This project explores gameplay using reinforcement learning in Atari 2600 Pac-Man, one of the most well-known Atari games. The project focuses on developing and evaluating reinforcement learning agents capable of mastering the game, providing insights into learning strategies and challenges within this gaming environment. Proximal Policy Optimization combined with Generalized Advantage Estimation enabled stable and efficient policy learning over 30,000 episodes and 24.6 million timesteps, achieving final evaluation rewards exceeding 500 points. Despite strong training improvements, minor reward gains during evaluation indicate challenges in policy generalization. Beyond gaming, PPO has demonstrated effectiveness in fine-tuning large language models, such as ChatGPT, via reinforcement learning from human feedback (RLHF), and it can be extended to multi-agent scenarios through algorithms like MAPPO. Future work will explore alternative algorithms, improved exploration and reward strategies, hybrid state representations, and robust evaluation methods to enhance learning efficiency and policy robustness in both single-agent and multi-agent domains.

1 Introduction

This project explores the application of reinforcement learning to sequential decision-making problems within the Pac-Man environment. The approach employs Proximal Policy Optimization (PPO) combined with Generalized Advantage Estimation (GAE), utilizing deep neural networks to approximate both the policy and value functions (Schulman et al., 2017). The implementation is performed in the `ALE/Pacman-v5` environment from the Arcade Learning Environment (Bellemare et al., 2012), where the learning agent aims to maximize the cumulative game score through strategic control.

Proximal Policy Optimization (PPO) was selected due to its empirical stability during training and its balance between implementation simplicity and performance compared to alternatives such as TRPO, TD3, and DDPG (Schulman et al., 2017). Beyond its success in game-based benchmarks, PPO has demonstrated effectiveness in diverse domains, including the alignment and fine-tuning of large language models like ChatGPT (Ouyang et al., 2022; Unsloth, 2025), as well as continuous control tasks in environments such as Lunar Lander and AWS DeepRacer simulations (Foundation, 2025; Balaji et al., 2019).

Extensions of PPO have been introduced for multi-agent settings, most notably through Multi-Agent Proximal Policy Optimization (MAPPO), which has achieved strong results in cooperative environments such as the Overcooked AI challenge (Carroll et al., 2019). MAPPO addresses challenges including non-stationarity and coordination by incorporating centralized training with decentralized execution, outperforming independent PPO and Multi-Agent Deep Deterministic Policy Gradient in stability, scalability, and sample efficiency (Yu et al., 2021). Although MAPPO is beyond the scope of this study, its success highlights the adaptability of PPO to complex multi-agent environments.

The scope of this project is limited to the standard Pac-Man environment, in contrast to Ms. Pac-Man, which was tested in the original PPO evaluation. This project includes problem formulation, algorithmic design, neural network architecture, training methodology, and empirical evaluation, concluding with key observations and directions for future work.

2 Pac-Man Environment

The Arcade Learning Environment (ALE) provides a standardized interface for interacting with Atari 2600 games, transforming them into well-defined reinforcement learning tasks (Bellemare et al., 2012). This project focuses on the `ALE/Pacman-v5` environment, which emulates the 1982 Atari 2600 version of the classic arcade game *Pac-Man* (Farama Foundation, 2025).

2.1 Environment Description

The `ALE/Pacman-v5` environment challenges the agent to navigate a maze, avoid ghosts, and collect pellets to maximize cumulative score. The game mechanics are consistent with the original design, providing a dynamic setting for sequential decision-making.

2.2 State Space

Atari environments offer three observation modes: RGB, grayscale, and RAM. RGB provides a $210 \times 160 \times 3$ color image, grayscale a 210×160 monochrome image, and RAM a compact 128-dimensional vector representing the console's internal memory. Traditional deep reinforcement learning approaches typically rely on raw pixel inputs processed by convolutional neural networks (CNNs). In contrast, this project leverages the ALE's internal state via the 128-byte RAM vector, which reduces the overhead of visual processing, reduces computational cost, and provides direct access to symbolic game variables. This enables a more focused evaluation of decision-making strategies. Each RAM byte (0–255) corresponds to a memory address and, unlike image inputs, lacks explicit spatial structure, requiring the agent to infer relationships purely from symbolic state data. Key memory locations encode Pac-Man's position, ghost positions, pellet states, power-up timers, score, and lives. Due to limited official documentation, these mappings are largely derived from community-driven reverse engineering.

2.3 Action Space

The agent interacts with the Pac-Man environment through a discrete action space of five actions. Although the Atari 2600 controller supports 18 actions, this reduced set is sufficient for gameplay and serves as the default in the Arcade Learning Environment. The actions correspond to the four cardinal directions and a no-operation (NOOP) option, defined as:

$$A = \{\text{NOOP}, \text{UP}, \text{RIGHT}, \text{LEFT}, \text{DOWN}\}$$

2.4 Rewards and Termination Condition

The reward structure is adapted from Pac-Man's game scoring rules, defined as follows:

- Video wafers: +1 point.
- Power pills: +5 points, granting temporary ability to eat ghosts.
- Vitamins: +100 points when collected.
- Ghosts: Points scale sequentially during a power-pill effect (20, 40, 80, 160 points).
- Reward shaping (New): A penalty of -0.1 points per time step to encourage efficient play.

An episode terminates when all four lives are lost or the maze is cleared.

2.5 Challenge and Complexity

Pac-Man presents a challenging reinforcement learning environment due to its combination of sequential planning, adversarial dynamics, and uncertainty. The agent must perform real-time navigation while adapting to ghosts with distinct behavioral patterns. The game also incorporates stochasticity through sticky actions and randomized ghost movements, demanding policies that are robust to variability. These factors collectively make Pac-Man a rigorous benchmark for evaluating reinforcement learning algorithms, especially in developing strategies that balance exploration, adaptability, and efficiency under temporal constraints.

3 Proximal Policy Optimization with GAE

Clipped Proximal Policy Optimization (PPO) combined with Generalized Advantage Estimation (GAE) offers a robust and efficient approach to policy learning. The clipped surrogate objective, together with GAE, prevents large policy updates that could destabilize training, resulting in more reliable performance and easier hyperparameter tuning.

3.1 Actor-Critic Architecture

PPO utilizes an actor-critic framework composed of two neural networks trained concurrently:

- **Actor (Policy Network, π_θ)**: This network models the policy by mapping the current state s_t to action probabilities. It consists of a four-layer multilayer perceptron (MLP) with ReLU activations implemented in PyTorch. The output layer produces raw action logits without activation, facilitating integration with categorical distributions.
- **Critic (Value Network, V_ϕ)**: This network estimates the state value $V_\phi(s_t)$, representing expected cumulative future reward from state s_t. The critic shares the same architecture as the actor, including orthogonal weight initialization, and outputs a single scalar without nonlinear activation.

3.2 Network Architecture for RAM-Based Inputs

Given the 128-dimensional Atari RAM state as input, a multilayer perceptron (MLP) architecture was employed for both actor and critic, as discussed in Section 3.1. This approach contrasts with pixel-based methods that typically use convolutional neural networks to capture spatial features.

Each network consists of four fully connected layers with 256 hidden units per layer and ReLU activations. Layer Normalization is applied after each hidden layer to improve training stability by reducing internal covariate shift. The actor network outputs logits corresponding to the five discrete actions available in the environment, while the critic outputs a scalar value estimate.

3.3 Clipped Surrogate Objective

The core innovation in PPO is the clipped surrogate objective, which limits policy updates by constraining the probability ratio between new and old policies, thereby enhancing training stability and efficiency:

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

The objective function is defined as:

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

Here, $\hat{A}_t$ denotes the estimated advantage and ϵ defines the clipping threshold. This mechanism restricts excessively large policy updates while avoiding the complexity of trust region methods such as TRPO. The pseudocode for PPO implementation is illustrated in Figure 1.

Algorithm 1 PPO-Clip

Input: initial policy parameters θ_0, initial value function parameters ϕ_0
for $k = 0, 1, 2, \ldots$ **do**
  Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
  Compute rewards-to-go $\hat{R}_t$.
  Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.
  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.
  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.
end for

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

3.4 Generalized Advantage Estimation

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 one-step temporal difference error is defined as:

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

which has low variance but potential bias due to dependence on value estimates. Monte Carlo returns provide unbiased estimates but suffer from high variance.

GAE computes an exponentially weighted sum of temporal difference errors:

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

where λ controls the bias-variance trade-off. Values close to 0.95 have shown empirical effectiveness. This approach yields smoother, more informative advantage signals, complementing PPO's clipped updates for improved learning stability.

3.5 Rollout Buffer

The RolloutBuffer collects and stores experience tuples during training, including states, actions, action log probabilities, rewards, done flags, and value estimates at each time step. Accumulated data over rollouts enables multiple minibatch epochs per update, enhancing sample efficiency and stabilizing policy optimization (Brunskill et al., 2025; OpenAI, 2017).

3.6 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 Training Configuration

This section details the training configuration. Hyperparameters were selected following established best practices from prior work, including PPO implementations on the Lunar Lander environment and MAPPO applied to the Overcooked AI task (Foundation, 2025; Balaji et al., 2019), ensuring consistency and comparability in performance evaluation.

4.1 Hyperparameter Overview

The performance of the PPO agent in the `ALE/Pacman-v5` environment is highly sensitive to the tuning of key hyperparameters. These parameters critically influence training dynamics, including the stability of policy updates, the balance between exploration and exploitation, and overall sample efficiency. The values presented in Table 1 were selected based on established literature and prior practical experience, then further refined through empirical evaluation. This configuration facilitates stable convergence of the actor-critic architecture while ensuring robustness throughout training.

4.2 Hyperparameter Configuration

Table 1 summarizes the key hyperparameters used during training, including learning rates, discount factors, PPO clipping thresholds, network architecture parameters, and scheduling intervals for checkpointing and result saving.

Hyperparameter	Symbol	Description	Value
Learning rate (actor)	α_{actor}	Actor network learning rate	2.5×10^{-4}
Learning rate (critic)	α_{critic}	Critic network learning rate	5×10^{-4}
Discount factor	γ	Reward discounting factor	0.99
Epochs per update	K	Policy update iterations per batch	10
Clipping parameter	ϵ	PPO objective clipping threshold	0.2
GAE parameter	λ	Controls bias-variance trade-off	0.96
Hidden layer size	h	Units per hidden layer in networks	256
Maximum timesteps	$T_{\max}$	Maximum steps per episode	20,000
Update interval	T_{update}	Timesteps between policy updates	8,192
Checkpoint frequency	-	Episodes between model checkpoint saves	5,000

Table 1: Hyperparameters employed in PPO training

5 Experimental Results and Analysis

The agent was trained for a total of 24.5 million timesteps across 30,000 episodes, requiring 14.4 hours in the `ALE/Pacman-v5` environment using RAM-based observations. This section presents a detailed analysis of the agent's learning progression, qualitative evaluation of emergent behaviors, and interpretation of policy decisions linked to the underlying RAM state.

Table 2 summarizes model performance across training episodes on a Colab L4 GPU.

Episodes	Time (min)	Cumulative Time (hr)	Cumulative Steps	Avg Reward (Last 100 eps)	Avg Reward (Unshaped)	Eval Reward (100 eps)
1–5000	126.27	2.10	3,590,291	66.60	138.40	23.93
5001–10000	131.67	4.30	7,359,393	197.66	273.05	344.82
10001–15000	144.32	6.70	11,482,844	286.32	368.79	409.53
15001–20000	155.22	9.29	15,879,247	337.27	425.20	381.54
20001–25000	157.55	11.92	20,276,052	377.02	464.96	518.19
25001–30000	151.42	14.44	24,511,479	411.19	495.90	506.12

Table 2: Summary of Training Progress and Evaluation Rewards

5.1 Training Progression and Learning Phases

The training process over 30,000 episodes can be divided into six phases, each characterized by distinct behavioral patterns, performance trends, and algorithmic dynamics, as shown in Figure 2.

Phase 1: Random Exploration (Episodes 1 to 5,000) During this initial stage, the policy behaves nearly randomly, frequently resulting in collisions with ghosts and entrapment. This is reflected in low average rewards of 66.6 (shaped) and 138.4 (unshaped), with an evaluation reward of 23.93. High entropy regularization encourages broad exploration; however, no meaningful action-reward associations are yet established. This exploration phase is critical for later training, as it facilitates the discovery of actions that yield high rewards.

Phase 2: Rapid Policy Acquisition (Episodes 5,001 to 10,000) During this phase, the agent learns fundamental strategies including ghost avoidance and pellet collection. Average rewards increase sharply to 197.7 (shaped) and 273.1 (unshaped), while evaluation rewards reach 344.8. This period is marked by strong advantage estimates driving substantial policy improvements.

Phase 3: Strategic Consolidation (Episodes 10,001 to 15,000) Navigation becomes more structured and efficient as the agent clears board sections methodically. Average rewards increase to 286.3 (shaped) and 368.8 (unshaped), with evaluation reward stabilizing near 409.5. PPO updates are moderate, reinforcing effective sequential behaviors.

Phase 4: Advanced Adaptation (Episodes 15,001 to 20,000) The agent adopts higher-level tactics such as strategic power pellet use. Average rewards rise to 337.3 (shaped) and 425.2 (unshaped), while evaluation reward decreases slightly to 381.5, indicating occasional failed experiments and potential overfitting. The value function emphasizes long-term gains over immediate rewards.

Phase 5: Fine-Grained Optimization (Episodes 20,001 to 25,000) Complex strategies emerge, including ghost herding and route optimization. Improvement slows as performance peaks with average rewards of 377.0 (shaped) and 465.0 (unshaped), and evaluation reward reaching 518.2. PPO clipping constrains updates, promoting refined improvements.

Phase 6: Performance Plateau (Episodes 25,001 to 30,000) Agent behavior stabilizes, leading to a performance plateau. Average rewards level at 411.2 (shaped) and 495.9 (unshaped),

while evaluation rewards slightly decline to 506.1. This indicates convergence to a local optimum, implying further improvements may need alternative exploration strategies or algorithmic changes.

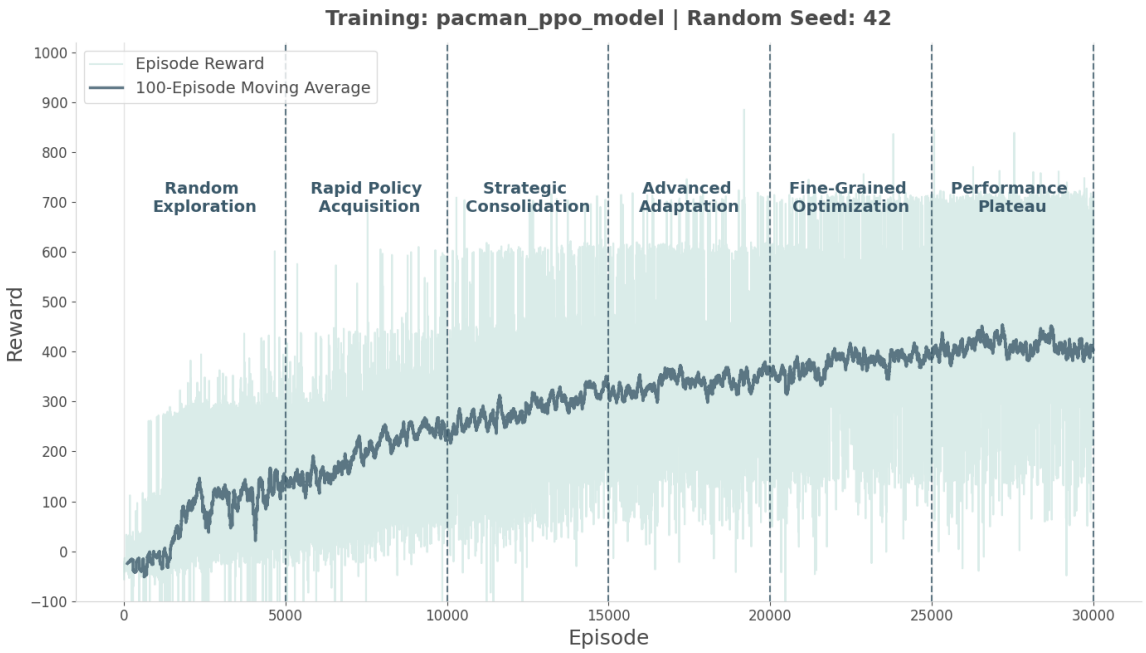

Figure 2: Average reward per episode with a 100-episode moving average over 30,000 episodes, showing learning progression from exploration to convergence.

5.2 Final Model Evaluation

The final PPO policy was evaluated over 100 episodes to assess generalization. Training rewards without shaping reached 495.9 by episode 30,000, with evaluation rewards averaging 506.12, indicating stable performance. The highest recorded evaluation score was 812 points, as shown in Figure 3. Although some low-value wafers remain, each worth only one point, the agent typically avoids collecting them. Instead, it takes risks to consume the vitamin, which is worth 100 points. This behavior suggests that appropriate reward shaping could better guide the agent to clear the maze when that is the desired objective.

Despite training rewards increasing between episodes 25,000 and 30,000, evaluation rewards slightly declined from 518.19 to 506.12, suggesting potential overfitting and limited generalization. Qualitative analysis confirms effective power pellet use, ghost avoidance, and pellet collection, though occasional failures in complex scenarios highlight areas for improvement. Overall, results demonstrate strong task proficiency and validate PPO's suitability for medium-complexity environments while underscoring challenges in sustained generalization.

6 Discussion and Conclusion

6.1 Main Takeaways

This project presents a comprehensive evaluation of a PPO agent trained with RAM-based state inputs in the Atari 2600 Pac-Man environment. Over 30,000 episodes on a Colab L4 GPU, the agent progressively developed strategic behaviors within a symbolic, non-spatial state space. The results provide valuable insights into the strengths and limitations of PPO in this gameplay environment.

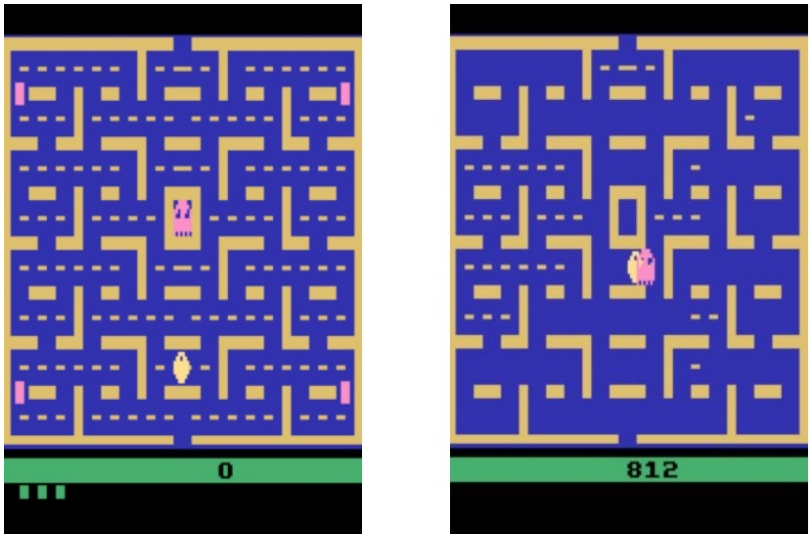

Figure 3: Final PPO model evaluation after 30,000 episodes, with a top score of 812 points.

Training Dynamics and Reward Shaping The phased learning progression underscores the critical role of reward shaping. Early-stage high-entropy exploration facilitated broad coverage of the state space, while shaped rewards effectively guided the agent toward efficient pellet collection and ghost avoidance. However, occasional declines in evaluation rewards despite improved training performance indicate potential overfitting to the shaped reward signals.

Exploration versus Exploitation Effective training required a careful balance between exploration and exploitation. The clipped policy updates in PPO helped moderate learning and contributed to the stabilization of complex, multi-step strategies including ghost herding and power pellet utilization. Additionally, the observed plateau in performance toward the end of training indicates that further gains may depend on the incorporation of novel exploration techniques or modifications to the environment.

Generalization and Performance Stability Evaluation across 100 episodes demonstrated stable policy performance with maximum scores exceeding 800 points. Slight decreases in evaluation scores near the end of training suggest that the learned policy may not fully generalize beyond the training environment. This highlights the potential benefit of more robust techniques, including mechanism design and reward shaping.

6.2 Future Work and Conclusion

Algorithmic and Architectural Improvements Exploring alternative reinforcement learning methods such as A2C or SAC, and combining RAM inputs with visual observations, could improve learning stability and better capture spatial information critical for decision making.

Exploration and Reward Design Incorporating intrinsic motivation or curiosity-driven exploration alongside refined reward structures, including bonuses for strategic actions, may enhance discovery of important game dynamics and accelerate skill acquisition.

Robust Evaluation and Generalization Future research should utilize multiple random seeds, varied game layouts, and difficulty settings to strengthen performance reliability and assess policy generalization across diverse scenarios.

References

Bharathan Balaji, Sunil Mallya, Sahika Genc, Saurabh Gupta, Leo Dirac, Vineet Khare, Gourav Roy, Tao Sun, Yunzhe Tao, Brian Townsend, Eddie Calleja, Sunil Muralidhara, and Dhanasekar Karuppasamy. Deepracer: Educational autonomous racing platform for experimentation with sim2real reinforcement learning. *arXiv preprint arXiv:1911.01562*, 2019. URL https://arxiv.org/abs/1911.01562.

Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *CoRR*, abs/1207.4708, 2012. URL https://arxiv.org/abs/1207.4708.

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.

Micah Carroll, Rohin Shah, Mark K. Ho, Thomas L. Griffiths, Sanjit A. Seshia, Pieter Abbeel, and Anca D. Dragan. On the utility of learning about humans for human-ai coordination. *arXiv preprint arXiv:1910.05789*, 2019. URL https://arxiv.org/abs/1910.05789.

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. Pacman environment — arcade learning environment. https://ale.farama.org/environments/pacman/, 2025. Accessed: 2025-08-01.

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.

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.

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

Chao Yu, Akash Velu, Eugene Vinitsky, Jiaxuan Gao, Yu Wang, Alexandre Bayen, and Yi Wu. The surprising effectiveness of ppo in cooperative, multi-agent games. *arXiv preprint arXiv:2103.01955*, 2021. URL https://arxiv.org/abs/2103.01955.