Reinforcement Learning with Proximal Policy Optimization for Strategic Betting in Counter-Strike 2 Esports

Nathan Ho
Drexel University
Philadelphia, PA, USA
nlh55@drexel.edu

Alexey Kuraev
Drexel University
Philadelphia, PA, USA
ak4249@drexel.edu

Matthew Protacio
Drexel University
Philadelphia, PA, USA
mp3634@drexel.edu

Abstract

This project explores the application of Proximal Policy Optimization (PPO), a reinforcement learning algorithm, to develop an intelligent betting agent for esports matches. We evaluated the agent's performance and decision making in a simulated betting environment using historical match data.

ACM Reference Format:
Nathan Ho, Alexey Kuraev, and Matthew Protacio. 2025. Reinforcement Learning with Proximal Policy Optimization for Strategic Betting in Counter-Strike 2 Esports. In *Proceedings of Drexel Research Project (Preprint).* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Sports betting is a widely practiced recreational activity in which individuals place bets on specific outcomes of sporting events. The types of bets range broadly, including predicting specific events during a game or determining the ultimate winner of a match. This paper specifically explores moneyline bets, where the bets are placed solely on the final result of a contest.

Despite its popularity, sports betting remains underrepresented as a quantitative research domain, partly due to its association with gambling, which contributes to limited academic inquiry and systematic study. Predicting winners accurately poses considerable challenges, as match outcomes are inherently stochastic due to significant variability in team performance and individual player dynamics.

Effective betting strategies often hinge on identifying edges, which involve detecting discrepancies between bookmaker odds and bettors' valuations. Precisely computing match odds from fundamental analyses demands extensive computational resources. However, by considering established bookmakers' odds as a reliable proxy for the market's perceived fair value, we circumvent extensive calculations while still gaining actionable insights.

Applying these concepts to professional esports introduces unique complexities. State representation in video games can lead to state explosion due to numerous exogenous variables. Moreover, continual game updates and modifications create unstable environments where strategies effective in one period may become obsolete in the next.

However, the esports title *Counter-Strike 2 (CS2)* offers specific advantages for quantitative modeling. Economic management significantly influences game-play outcomes, with the team's budget serving as a critical predictive indicator. Within *CS2*, team economics is determined by several well-defined factors, including purchases of weapons and equipment, the income from the wins and the earnings from the elimination of opponents. Matches typically span best-of-13 rounds, allowing for temporal economic analysis across discrete intervals.

This paper applies reinforcement learning, specifically, Proximal Policy Optimization (PPO), to leverage these economic indicators for betting decisions. We further explore reward function formulations beyond simple correctness, investigating the impact on rewarding expected returns based on betting odds. In doing so, we examine the limitations of traditional betting methodologies, exploring whether integrating financial and probabilistic principles yields improved betting outcomes in esports scenarios.

2 Methodology

2.1 Data Collection via Web Scraping

The dataset used to train the PPO agent was constructed by scraping esports match data from two primary sources: *https://bo3.gg* for detailed Counter-Strike 2 (CS2) match data and *https://www.oddsportal.com* for corresponding betting odds. The scraping process was implemented using Python scripts utilizing the libraries *Playwright* and *BeautifulSoup*.

Match Data (bo3.gg): Utilized *Playwright* for automated browser control to navigate and load dynamically-rendered web pages, ensuring complete data retrieval. Extracted JSON data containing round-level statistics, including economic state, round results, map data, player statistics, and team performance metrics.

Betting Odds Data (oddsportal.com): Leveraged *Playwright* to handle interactive and dynamically updated odds information, which required simulating user interaction to reveal hidden or paginated odds. Employed *BeautifulSoup* to parse HTML content efficiently, extracting structured odds including opening, closing, and intermediate odds offered by various bookmakers.

The scraping scripts were developed with careful adherence to ethical web-scraping practices, employing randomized delays and respectful request frequencies to avoid overwhelming the servers. Additionally, extracted data was systematically stored in JSON files for ease of processing and reproducibility. The final compiled dataset provided a comprehensive representation of match states

Preprint, Philadelphia, PA
© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

and betting odds, enabling robust feature extraction for the PPO model training pipeline. Scraped data was later analyzed for feature distribution and building a dictionary of winners for all games found.

2.2 Feature Engineering

After web scraping, a large collection of JSON formatted match rounds is accumulated. While raw economic indicators provide foundational insight, their direct application can suffer from excessive noise and limited predictive value.

To enhance signal strength, we compute more sophisticated financial metrics. By modeling a team as a dynamic market asset, we can track and aggregate performance indicators over time, offering temporal context that enhances the predictive power of our features.

An example of the raw JSON match data structure is shown below:

```
{
  "match_id": "furia-vs-mibr-12-05-2025",
  "tournament": "PGL Astana 2025",
  "team_a": "FURIA",
  "team_b": "MIBR",
  "status": "Ended",
  "game_count": 3,
  "games": [
    {
      "game_index": 1,
      "map": "train",
      "rounds": [
        {
          "round_number": 1,
          "initial_team_a_econ": 4000,
          "initial_team_b_econ": 4000,
          "buy_team_a": "eco",
          "buy_team_b": "full",
          "final_team_a_econ": 3600,
          "final_team_b_econ": 4200,
          "round_winner": "team_b"
        },
        ...
      ]
    },
    ...
  ]
}
```

The raw JSON match data is preprocessed to serve as a state input for the PPO model. Each round in a game is transformed into a normalized *PyTorch* tensor. This ensures efficient and stable model training. As mentioned above, we aimed to convert raw team economic status into meaningful signals. These economic metrics help to capture team performance dynamics. These are defined as follows:

Delta Econ for Both Teams - Captures the absolute economic change between the starting and ending bankroll of a team within a round:

$$\Delta\text{Econ}_T = \text{Final Economy}_T - \text{Initial Economy}_T, \quad T \in \{\text{A}, \text{B}\} \tag{1}$$

where $T \in \{\text{A}, \text{B}\}$ denotes the team.

ROI based on Team Econ - Measures the relative financial gain or loss by comparing final economic status to initial investment:

$$\text{ROI}_T = \frac{\text{Final Economy}_T - \text{Initial Economy}_T}{\text{Initial Economic Value}_T}, \quad T \in \{\text{A}, \text{B}\} \tag{2}$$

Odds-Based Return on Investment (Odds ROI) – Estimates the expected return based on bookmaker odds, defined for each team as:

$$\text{Odds ROI}_T = (\text{DecimalOdds}_T \times \text{WinProb}_T) - 1, \quad T \in \{\text{A}, \text{B}\} \tag{3}$$

Implied Probability from Odds – Represents the bookmaker's implicit estimation of each team's chance of winning, computed as:

$$\text{ImpliedProb}_T = \frac{1}{\text{DecimalOdds}_T}, \quad T \in \{\text{A}, \text{B}\} \tag{4}$$

Cost Per Kill (CPK) - Quantifies a team's economic efficiency regarding combat effectiveness:

$$\text{CPK}_T = \frac{\text{Economic Investment per Round}_T}{\text{Number of Kills per Round}_T}, \quad T \in \{\text{A}, \text{B}\} \tag{5}$$

where Economic Investment per Round denotes the team's purchase amount during buy phase.

Expected Value (EV) for a Bet – Represents the average outcome an agent can expect over time, based on the probability of winning and associated gains/losses:

$$\text{EV} = P \cdot \text{Profit} + (1 - P) \cdot \text{Loss} \tag{6}$$

where P is the estimated probability of winning the bet, and Profit and Loss are the respective monetary outcomes.

Kelly Criterion – A formula introduced by J.L. Kelly [4], used to determine the optimal fraction of an agent's bankroll to wager in order to maximize long-term growth:

$$f^* = \frac{bp - q}{b} \tag{7}$$

where f^* is the optimal bet fraction, b is the net odds in decimal form (i.e., odds − 1), p is the estimated probability of winning, and $q = 1 - p$ is the probability of losing.

The following are game features that did not require

$$\text{Score}_T, \quad T \in \{\text{A}, \text{B}\} \tag{8}$$

Score of the current round state. Data is sampled as a string (e.g. "1-5") and split into respective team score for the round.

$$\text{Kills}_T, \quad T \in \{\text{A}, \text{B}\} \tag{9}$$

Kills for each team for current round state.

$$\text{Duration}_{round}, \quad round \in \{1, \text{Terminal Round}\} \tag{10}$$

Duration of the round in seconds.

The raw JSON match data obtained required preprocessing to serve as input for the Proximal Policy Optimization (PPO) model. Each round in a game was transformed into normalized PyTorch tensors, ensuring efficient and stable model training. Numerical features, such as team economic status, round outcomes, and individual player performance statistics, were normalized using min-max scaling or standardization to ensure consistency across varying scales.

Features were selected based on their predictive value regarding match outcomes and their ability to encapsulate meaningful temporal and economic context. Specifically, selected features included team bankroll, Return on Investment (ROI), implied probability derived from betting odds, ROI based on odds, and Cost Per Kill (CPK). This targeted selection aimed to reduce dimensionality, improve signal-to-noise ratios, and enhance the model's capacity to generalize from historical data to future betting scenarios.

2.3 Normalization of Features

An important process before sending states to our PPO input model is to normalize our findings. Feature normalization is the process of transforming input features to a common scale or range. This ensures that data with large numerical ranges do not dominate smaller ranges during training. This preprocess helps models converge faster and learn more stable representation.

Finding maximum values was easy for certain game stats, as *Counter-Strike 2* is heavily documented with wikis and blogs. The maximum amount of players per team in a professional match is five players [1].

Normalizing player elimination is as follows:

$$\text{Kills}_{T,\,\text{norm}} = \frac{\text{Kills}_T}{5}, \quad T \in \{\text{A}, \text{B}\} \tag{11}$$

The maximum amount a player can hold is $16,000 [2]. Thus, for five players, that equates to a max of $80,000 per team.

Normalization of delta economic value is as follows:

$$\Delta\text{Econ}_{T,\text{norm}} = \frac{\Delta\text{Econ}_T}{80{,}000}, \quad T \in \{\text{A}, \text{B}\} \tag{12}$$

Not all features have clearly defined or bounded maximum values, particularly those representing continuous variables such as round duration, rounds in a match, or ROI. As a result, we could not rely solely on predefined constants for normalization. Instead, we analyzed each feature's distribution — the way values are spread across the dataset — to better understand its central tendency and variance. Examining the distribution allowed us to identify appropriate clipping thresholds to detect outliers. Using these, we select scaling methods such as log transformations or z-score normalization. This preserved the integrity of the data while improving model stability.

To determine an appropriate upper bound for round duration, we plotted a histogram with a marker at the 95th percentile. While the nominal maximum duration of a round is 1 minute and 55 seconds, with an additional 40 seconds possible after a bomb plant, we observed that round times exhibit stochastic behavior. Factors such as technical pauses, timeouts, and stalled rounds contribute to a continuous and skewed distribution. These rare cases disproportionately affect the feature distribution and can negatively impact model training. By capping round duration at the 95th percentile, we retain the majority of meaningful observations while enabling more stable normalization and feature scaling.

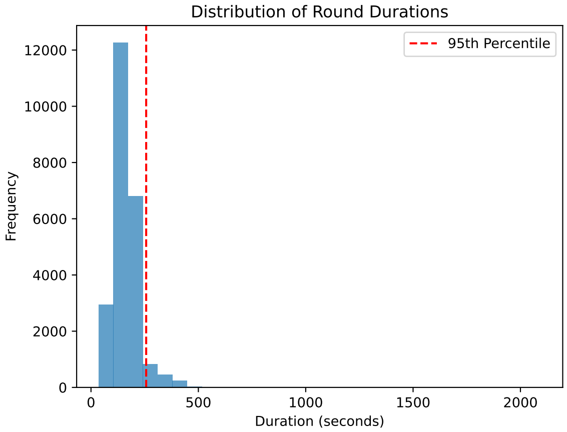

Figure 1: Distribution of round durations across 464 CS2 matches. A red dashed line marks the 95th percentile, which is used to cap this feature during normalization.

The score is used as a feature for our input vector. This is normalized as such:

$$\text{Score}_{T,norm} = \frac{\text{Score}_T}{\text{Max Rounds Possible}}, \quad T \in \{\text{A}, \text{B}\} \tag{13}$$

However, round number proved to be another feature that needed appropriate bound handling. In *CS2* rounds can exceed past the twelve round if teams tie, leading into overtime. In theory, *CS2* matches could have infinite rounds. In order to appropriately scale round numbers, we observed the 95th percentile across all accumulated match data.

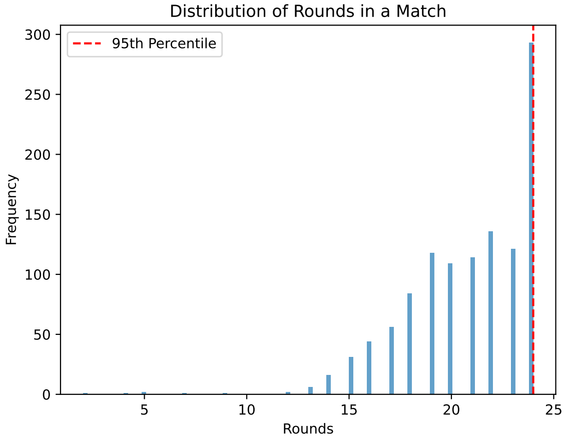

Figure 2: Distribution of total rounds of a match across 464 CS2 matches. A red dashed line marks the 95th percentile, which is used to cap this feature during normalization.

Our dataset does not include overtime rounds, which are common in closely contested matches that exceed the standard 24-round regulation period. This introduces a potential limitation in training stability, as the agent may develop an implicit bias against matches with extended durations or fail to learn strategies that are relevant in overtime contexts.

While acknowledging this gap, we chose to exclude overtime rounds due to time constraints and the added complexity of scraping and preprocessing additional match data. Incorporating overtime rounds remains a promising direction for future work, as doing so would allow for more robust agent generalization and greater realism in match simulation.

We also examined the distribution of a team's round level ROI. To visualize the spread across teams, we used a violin plot to highlight both the density and variance in ROI outcomes.

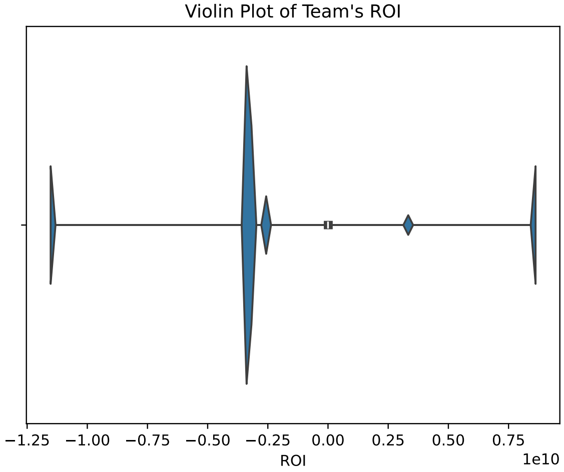

Figure 3: Distribution of a team's ROI based on their economic performance.

The distribution is highly skewed, with a small number of rounds producing extreme positive or negative values. To stabilize this feature for training, we cap ROI at the 99th percentile with an absolute value of 24.0 and apply a signed log transformation. This preserves information while compressing extreme magnitudes, improving stability and preventing outlier dominance during learning.

Lasty, in order to encode the match outcome, we apply one-hot encoding to the winner label, resulting in a two-dimensional binary vector indicating whether Team A or Team B won [?]. This representation avoids introducing ordinal bias into the model and ensures compatibility with neural network inputs."

$$\text{Team A win} \longrightarrow [1,\ 0]$$
$$\text{Team B win} \longrightarrow [0,\ 1]$$

3 PPO Model Overview

PPO is a policy gradient reinforcement learning algorithm. PPO is inteneded to optimize the policy performance while maintaining training stability. This algorithm was developed by researchers at Open AI, ihntroduced in 2017 by Schulman et al [5]. This was a simpler alternative to Trust Region Policy Optimization (TRPO).

Unlike TRPO, which relied on complex second-order optimization, PPO uses a clipped surrogate objective that restricts policy updates to stay within a safe range. The clipping mechanism helps stabilize learning by preventing excessive policy shifts, which risk training stability. PPO is favored for its ease of implementation and sample efficiency. PPO is also noted for strong empirical performance across continuous and discrete action space [5].

Our project employs PPO to train a betting agent that makes decisions based on game features described above. While using strong market financial signals, we aimed to evaluate perfomance comparing against agents using various reward and action spaces.

3.1 Model Choice

For our PPO structure, we used a popular model found on *GitHub* developed by Nikhil Barhate [3]. In their implementation, the actor-critic use a shared network structure. Initial layers of the unified neural network process input states and then split into two separate heads: one for the actor and another for the critic. This allows for both the policy and value function to share common layers, reducing redundancy and computation overhead.

To guide policy and value function updates, the advantage function is estimated using Monte Carlor returns. For each time step in an episode, the agent computes the cumulative discounted reward based on the full trajectory. This serves as an estimate for how favorable a given state-action pair was compared to the baseline value function. While this method introduces higher variance compared to bootstrapped alternatives like Generalized Advantage Estimation (GAE), this advantage estimation provides a straightforward way to compute advantages from complete episode data.

3.2 Hyperparameters

Learning Rates: Discount Factor: Clipping Parameter: Batch Size: Epochs:

4 Training Procedure

Our project aims to evaluate multiple agent configurations by varying reward functions and action spaces. The choice of reward function plays a critical role in shaping agent behavior, as poorly designed rewards can hinder effective learning.

While betting naturally lends itself to a continuous action space, allowing for flexible wager sizes, we constrain the action space to a discrete set of options for training simplicity and stability.

4.1 Reward Function Exploration

Reward function is calculated as:

$$r : (\text{action},\ \text{outcome}) \longrightarrow \mathbb{R}$$

Our basic reward function is defined as:

$$r_{\text{basic}} = \begin{cases} +1, & \text{if bet is correct} \\ -1, & \text{if bet is incorrect} \\ 0, & \text{if no bet is placed} \end{cases} \tag{14}$$

Our complex reward function is defined as:

$$r_{\text{complex}} = \begin{cases} \text{Stake} \times (\text{Odds} - 1), & \text{if bet is correct} \\ -\text{Stake}, & \text{if bet is incorrect} \\ 0, & \text{if no bet is placed} \end{cases} \tag{15}$$

4.2 Action Space Definitions

A basic action space is defined as a discrete space of three actions: abstaining, betting on team A, or betting on team B.

$$\mathcal{A}_{\text{basic}} = \{\text{abstain}, \text{bet A}, \text{bet B}\} \tag{16}$$

A complex action space is defined as a discrete space of nine actions: abstaining, betting $\{5, 10, 25, 50\}$ percent of agent's bankroll on either team A or B

$$\mathcal{A}_{\text{complex}} = \{\text{abstain}\} \cup \{(t, p) \mid t \in \{\text{A}, \text{B}\},\ p \in \{5\%, 10\%, 25\%, 50\%\}\} \tag{17}$$

With two different reward functions and action spaces, our project compares the perfomance of three different agents using various combinations. These three agents are defined as so:

Table 1: Agent Comparison Across Reward Functions

Agent	Basic Reward	Complex Reward
Agent 1 (no bankroll)	✓	
Agent 2 (with bankroll)		✓
Agent 3 (with bankroll)		✓

Table 2: Agent Comparison Across Action Spaces

Agent	Basic Actions	Complex Actions
Agent 1 (no bankroll)	✓	
Agent 2 (with bankroll)	✓	
Agent 3 (with bankroll)		✓

5 Experiments and Results

6 Discussion

7 Conclusion

References

[1] n.d.. Competitive - Counter-Strike Wiki. https://counterstrike.fandom.com/wiki/Competitive. Accessed: 2025-06-02.

[2] n.d.. Money - Counter-Strike Wiki. https://counterstrike.fandom.com/wiki/Money. Accessed: 2025-06-02.

[3] Nikhil Barhate. 2020. PPO-PyTorch. https://github.com/nikhilbarhate99/PPO-PyTorch. Accessed: 2025-06-01.

[4] J. L. Kelly. 1956. A New Interpretation of Information Rate. *Bell System Technical Journal* 35, 4 (1956), 917–926. doi:10.1002/j.1538-7305.1956.tb03809.x

[5] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. (2017). arXiv:1707.06347 [cs.LG] https://arxiv.org/abs/1707.06347