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, 21 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 winner 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 game 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 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, 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 Counter-Strike 2 and Moneyline Betting

Counter-Strike 2 (CS2) is a competitive, team-based first-person shooter developed by Valve Corporation as the successor to Counter-Strike: Global Offensive. In a standard professional setting, two teams of five players compete in a best-of-24 round format, with each team alternating between offensive (Terrorists) and defensive (Counter-Terrorists) roles. The first team to win 13 rounds claims victory, with overtime played in the case of a 12-12 tie [1].

At the beginning of each round, players receive a monetary reward based on performance in the previous round, which can be used to purchase weapons and utility. These purchasing decisions lead to well-defined economic patterns known as *buy types*, which are categorized from low spending to high spending, explored later in this paper [2].

In esports betting, a common wager type is the moneyline bet, which requires predicting the outright winner of a match. Bookmakers assign odds to each team based on perceived win probabilities. For example, a bet on a +150 underdog yields a \$150 profit on a \$100 stake if the team wins.

3 Hypothesis

We hypothesize that incorporating sophisticated market signals in the input vector will enhance the agent's performance in learning profitable betting strategies. By integrating these rich representations of betting market dynamics and game-state context, the agent is expected to gain better long-term value and make more informed wagering decisions. We anticipate that models trained

ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

with high-level features will outperform those relying solely on raw game statistics or simplistic heurisitics.

4 Methodology

4.1 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*. Our states are defined as the end of a round in a match between Team A and Team B.

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 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. These odds are aligned with rounds of a match to accurately depict a full game state.

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 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.

4.2 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:

```json
{
  "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:

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

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

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

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

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

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

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

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

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

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

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 [6], 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 further calculations:

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

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 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.

4.3 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.

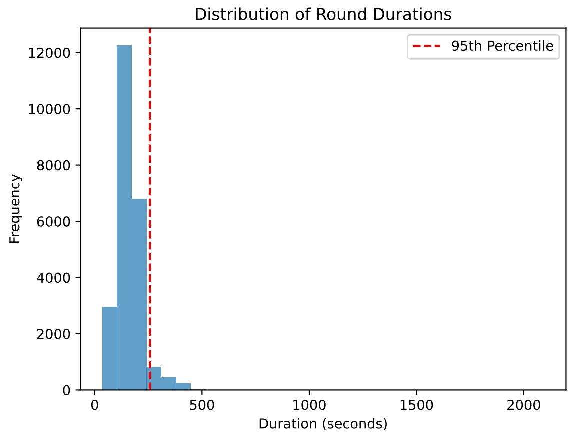

Figure 1: Distribution of round durations across 464 CS2 matches. A red dashed line marks the 95th percentile, which is used to cap this feature during normalization.

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.

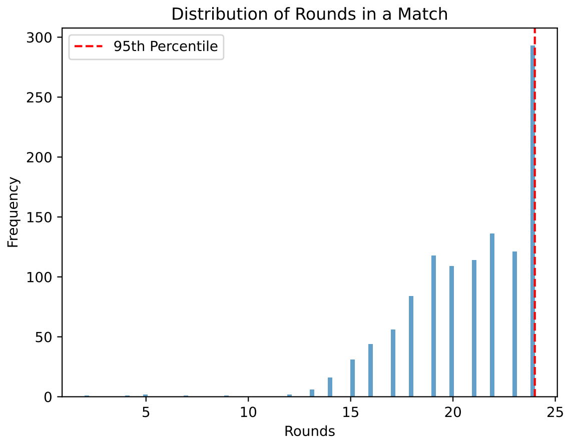

Figure 2: Distribution of total rounds of a match across 464 CS2 matches. A red dashed line marks the 95th percentile, which is used to cap this feature during normalization.

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.

As seen in Figure 3, 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.

Lastly, 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 [4]. This

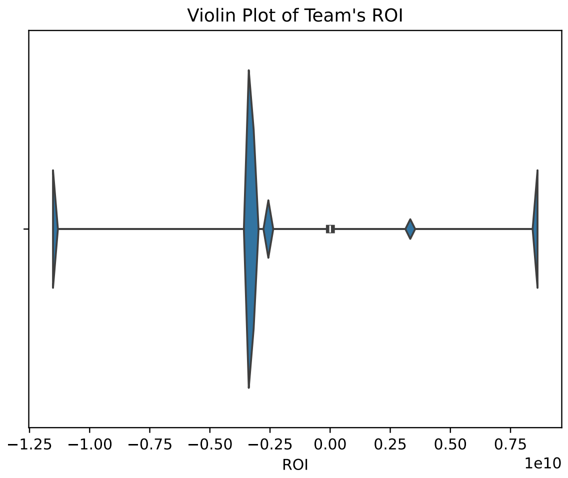

Figure 3: Distribution of a team's ROI based on their economic performance.

representation avoids introducing ordinal bias into the model and ensures compatibility with neural network inputs."

$$\begin{aligned} \text{Team A win} &\longrightarrow [1,\ 0] \\ \text{Team B win} &\longrightarrow [0,\ 1] \end{aligned}$$

4.4 Control Variable

In order to verify that rich market signals provided an advantage, we employed an *Agent 0* as our ground level experiment. This agent would not use any sophisticated signals and rely on the raw game state features from scraping. *Agent 0* includes the variables previously defined as such:

$$\text{Score}_T, \text{Kills}_T, \text{Duration}_{round} \tag{14}$$

Additionally, since the agent does not utilize advanced economic signals such as delta economy, expected value (EV), or the Kelly Criterion, we introduce a categorical feature known as the *buy type*. During the buy phase, each team allocates funds that can be categorized into one of four discrete tiers based on spending hierarchy: *eco*, *force*, *semi-buy*, and *full-buy*. These categories offer a simplified yet informative heuristic derived directly from game state data, without requiring external calculations.

For each team $T \in \{\text{A}, \text{B}\}$, buy types are one-hot encoded as follows:

$$\begin{aligned} \text{Buy Type}_T(\text{eco}) &= [1,\ 0,\ 0,\ 0,\ 0] \\ \text{Buy Type}_T(\text{force}) &= [0,\ 1,\ 0,\ 0,\ 0] \\ \text{Buy Type}_T(\text{semi-buy}) &= [0,\ 0,\ 1,\ 0,\ 0] \\ \text{Buy Type}_T(\text{full-buy}) &= [0,\ 0,\ 0,\ 1,\ 0] \\ \text{Buy Type}_T(\text{unknown}) &= [0,\ 0,\ 0,\ 0,\ 1] \end{aligned} \tag{15}$$

We include an additional *unknown* flag to account for instances where a team's buy type cannot be reliably determined. This preserves contextual integrity without introducing artificial bias into the model input.

5 PPO Model Overview

PPO is a policy gradient reinforcement learning algorithm. PPO is intended to optimize the policy performance while maintaining training stability. This algorithm was developed by researchers at Open AI, introduced in 2017 by Schulman et al [8]. 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 [8].

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 compared against agents using various reward and action spaces.

5.1 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.

5.2 Hyperparameters

We selected these hyperparameters based on values recommended in the original PPO paper [8], which have been shown to work well across a range of tasks. They offer a strong baseline for stable training without extensive tuning, allowing us to focus on adapting PPO to the betting domain.

Our hyper parameters are shown in Table 1. Agent 4 is an additional thought experiment to validate whether or not a non-discrete action space results to poor performance.

Table 1: PPO Hyperparameters

Hyperparameter	Value
Actor learning rate	0.0003
Critic learning rate	0.001
Discount factor (γ)	0.99
Epochs per update (K_{epochs})	80
Clipping parameter (ϵ_{clip})	0.2
Continuous action space	`false (except for Agent 4)`
Initial action std. dev.	0.6
Device	`device`

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.

During both training and testing, the agent could place or change a bet at the start of every round, using updated game state information before each decision. In reality, once you place a wager on a sportsbook, it is locked in for the entire match and cannot be adjusted mid-game; our simulation instead lets the agent freely revise its bet each round to learn from finer grained outcomes. This approach sacrifices realism, since real bets cannot be changed. This overestimate performance, but it was chosen to simplify implementation and fit our time scope.

6 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.

6.1 Feature Choice

We trained two versions of our PPO agent using different feature representations. The raw feature vector consists of unprocessed game state information (as previously described for Agent 0), directly reflecting the state of play. In contrast, the crafted feature vector includes additional market-derived signals such as odds, ROI, and betting indicators. This allows the agent to learn not just from gameplay data, but also from financial cues that influence real-world betting decisions.

Agents using a complex reward function must be assigned a bankroll, since the reward is based on the percentage of their stake. Unlike binary win or loss signals, this setup models profit or loss relative to how much the agent bets, making bankroll management an essential part of the learning process.

Table 2: Agent Comparison Across Feature Vector Choice

Agent	Raw Feature	Crafted Feature
Agent 0 (no bankroll/signals)	✓	
Agent 1 (no bankroll)	✓	
Agent 2 (w/ bankroll)		✓
Agent 3 (w/ bankroll)		✓

6.2 Reward Function Exploration

Reward function is calculated as:

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

Our basic binary reward function is defined as:

$$r_{\text{binary}} = \begin{cases} +1, & \text{if bet is correct} \\ -1, & \text{if bet is incorrect} \\ 0, & \text{if no bet is placed} \end{cases} \tag{16}$$

Our complex discrete reward function is defined as:

$$r_{\text{discrete}} = \begin{cases} \text{Stake} \times (\text{Odds} - 1), & \text{if bet is correct} \\ -\text{Stake}, & \text{if bet is incorrect} \\ 0, & \text{if no bet is placed} \end{cases} \tag{17}$$

A visual overview of our agent distribution across reward functions is as follows:

Table 3: Agent Comparison Across Reward Functions

Agent	Binary	Discrete
Agent 0 (no bankroll/signals)	✓	
Agent 1 (no bankroll)	✓	
Agent 2 (w/ bankroll)		✓
Agent 3 (w/ bankroll)		✓

6.3 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.

$$\mathcal{A}_{\text{basic}} = \{\text{abstain}, \text{bet A}, \text{bet B}\} \tag{18}$$

A complex discrete 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

$$\mathcal{A}_{\text{discrete}} = \{\text{abstain}\} \cup \{(t, p) \mid t \in \{\text{A}, \text{B}\},\ p \in \{5\%, 10\%, 25\%, 50\%\}\} \tag{19}$$

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 4: Agent Comparison Across Action Spaces

Agent	Basic	Discrete
Agent 0 (no bankroll/signals)	✓	
Agent 1 (no bankroll)	✓	
Agent 2 (with bankroll)	✓	
Agent 3 (with bankroll)		✓

6.4 Agent 4

We also benchmarked an agent that used a complex reward function and used a continuous action space. This was to demonstrate and experiment how a large non-discrete action space would performe against discrete actions.

Table 5: Agent Comparison Across Reward Functions

Agent	Complex R.	Continuous A.	Crafted
Agent 4 (w/ bankroll)	✓	✓	✓

7 Experiments and Results

To evaluate the training progress and behavior of our reinforcement learning agent, we log and visualize several key metrics throughout the training process.

For the following graphs, we denote the agents as such:

Table 6: Agent Legend Mapping

Agent	Legend Label
Agent 0	Binary Basic (Raw)
Agent 1	Binary Basic (Crafted)
Agent 2	Complex Discrete (Raw)
Agent 3	Complex Discrete (Crafted)
Agent 4	Complex Continuous (Crafted)

The **Cumulative Reward** graph illustrates the total reward accumulated by the agent over time, providing a high-level view of learning progression.

Early in training (up to roughly 20,000 steps), all agents begin near zero; the *Complex Continuous (Crafted)* agent (purple) rapidly climbs to a normalized reward of about 2 by 30,000 steps and then maintains a relatively stable plateau between 1.5 and 3, indicating that leveraging continuous profit signals alongside rich economic features helps the policy stabilize more quickly. In contrast, the *Binary Basic (Raw)* agent (orange) remains nearly flat (around −1) throughout training, demonstrating that a simple ±1 reward without economic context fails to guide meaningful learning. The *Binary Basic (Crafted)* agent (blue) shows modest improvement over the raw version, reaching normalized rewards near 1 by 100,000 steps, which suggests that even a basic reward can benefit from added economic features. The *Complex Discrete (Raw)* agent (red) exhibits high variance with large positive spikes (up to 5–6) but remains noisy and does not settle, indicating that a complex discrete reward alone can produce occasional large gains but lacks consistency. When the same discrete reward is combined with crafted features (*Complex Discrete (Crafted)*, green), the agent converges faster—reaching around 3 by 40,000 steps—before the training curve ends at 60,000 steps; this early convergence suggests that feature engineering reduces variance and accelerates learning in the discrete-reward setting. Overall, Figure 4 demonstrates that richer reward formulations and the inclusion of economic and market-derived features both improve convergence speed and lead to higher, more stable cumulative rewards compared to raw or binary reward functions alone.

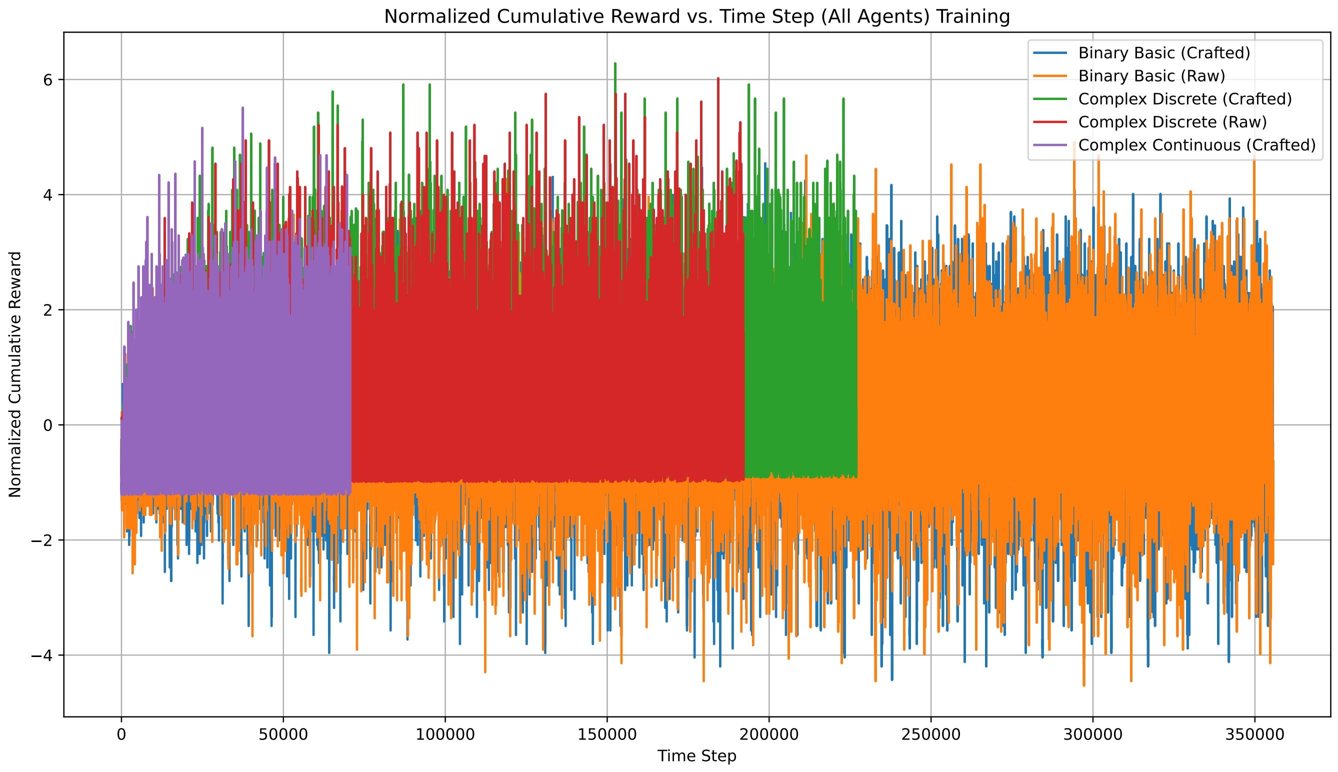

Figure 4: Cumulative reward during the training phase (all agents).

The **Policy and Value Loss** graph tracks the agent's optimization process, where decreasing policy loss indicates stabilization in policy updates, and decreasing value loss reflects improved estimation of expected returns.

Across all five agents, incorporating crafted features consistently leads to lower and more stable policy and value losses compared to raw-feature counterparts. In the binary reward setting, the Binary Basic (Raw) agent exhibits persistently high and noisy losses, whereas Binary Basic (Crafted) shows a gradual downward trend and reduced variance. For the Complex Discrete agents, the (Crafted) version converges most quickly—policy and value losses fall near zero by around episode 2000—while the raw variant remains erratic and slower to decrease. The Complex Continuous (Crafted) agent begins with higher initial loss but steadily declines thereafter, in contrast to its raw counterpart, which oscillates more throughout training. Overall, feature engineering not only accelerates convergence—especially for discrete rewards—but also dampens loss spikes, producing smoother learning curves.

The **Return Distribution** graph visualizes the spread of cumulative episode rewards; a tighter cluster around higher values suggests improved and consistent performance.

The return-distribution histograms show that agents with crafted features have much tighter, higher-centered reward clusters, whereas raw-feature agents display wide, low-centered spreads. For example, the *Binary Basic (Crafted)* agent's returns concentrate around modest positive values (mostly 0–60), while *Binary Basic (Raw)* has a scattered spread up to 600 with many near-zero outcomes . In the discrete-reward setting, *Complex Discrete (Crafted)* returns cluster densely in a high range (0–1.2×10), whereas *Complex Discrete (Raw)* shows a more erratic, lower-frequency spread up to 1.6×10. Finally, *Complex Continuous (Crafted)* produces a narrow spike below 100,000, indicating stable gains compared to its raw counterpart. Overall, engineered features lead to more consistent, centrally focused returns, while raw features result in extreme variance and lower median performance.

The **Bankroll** graph models the agent's financial trajectory over time, analogous to profit/loss curves in real-world trading or betting scenarios. A sustained upward trend indicates successful policy learning. Training is Figure **??** and testing is Figure **??**.

Agents using crafted features exhibit steadier and higher bankroll growth than the raw-feature agent in both training and testing. During training, the *Complex Discrete (Crafted)* agent's bankroll climbs smoothly to around $60,000 by 60,000 time steps, while the *Complex Discrete (Raw)* agent fluctuates more and stays lower. The *Complex Continuous (Crafted)* agent also outperforms the raw version but lags slightly behind the discrete crafted agent. In testing, the discrete crafted agent again ends with the highest balance,

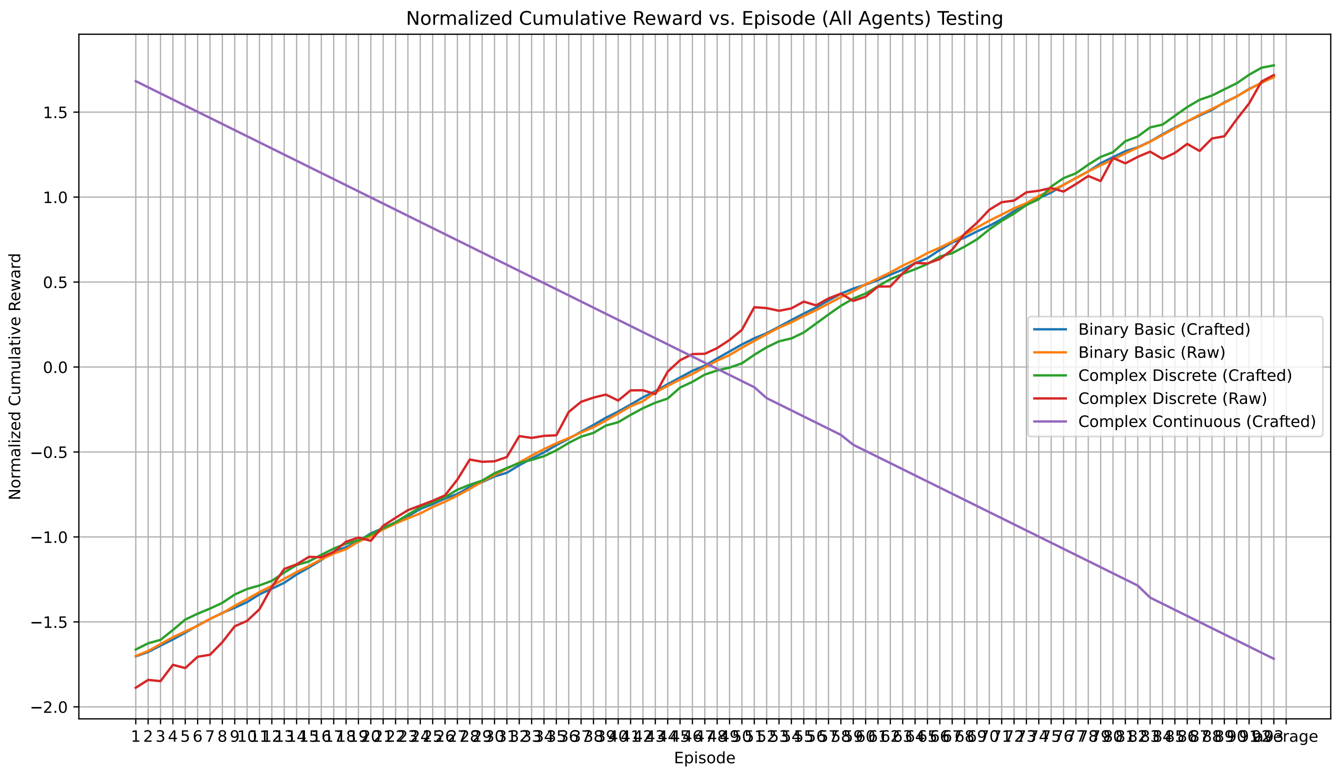

Figure 5: Cumulative reward during the testing phase (all agents).

followed by the continuous crafted agent, and the raw agent remains volatile with the lowest final bankroll.

Finally, the **Probability Calibration** graph compares the agent's predicted win probabilities against actual outcomes. Deviations from the diagonal reflect confidence misalignment—where curves above the diagonal indicate underconfidence, and below indicate overconfidence.

Figure 18 compares predicted win probabilities against empirical accuracies for all five agents. Both *Binary Basic (Crafted)* and *Binary Basic (Raw)* lie almost exactly on the diagonal, indicating near-perfect calibration at both 0.0 and 1.0 predicted probabilities. The *Complex Discrete (Crafted)* agent's points also cluster tightly around the diagonal (predictions at 0.05, 0.15, and 0.40 correspond to empirical accuracies near 1.0), suggesting well-aligned confidence estimates. In contrast, *Complex Discrete (Raw)* deviates substantially: a 0.10 predicted probability yields only 0.65 accuracy, and 0.18 yields 0.50, reflecting marked overconfidence. Finally, *Complex Continuous (Crafted)* almost exclusively predicts 0.0, with empirical accuracy likewise at 0.0—indicating systematic underconfidence. Overall, engineered discrete-reward agents achieve the closest alignment with perfect calibration, whereas raw and continuous-reward agents exhibit significant miscalibration :contentReference[oaicite:2]index=2.

8 Discussion

8.1 Limitations of PPO in Betting Scenarios

While PPO has proven effective in continuous control tasks, its application to sports betting context presents several challenges. Firstly, PPO assumes a relatively stable reward structure and a Markovian environment [8]. However, betting markets, especially in esports, are inherently non-stationary. Odds inevitably flunctuate rapidly based on real-time information. This can happen for various reasons, such as roster changes, game patch updates, and meta shifts. Consequently, the agent's policy is subject to deprecation between training iterations in a long term scope. Thus, without proper updates to the data and tweaks, we can expect suboptimal wagering decisions. Second, PPO's on policy nature requires fresh interaction with data for each update, which can be costly in real-money betting environments.

Gathering sufficiently diverse episodes without incurring large financial risk is difficult. Synthetic or simulated data may lack accurate reflection of live market dynamics [5]. PPO's gradient based update rules can struggle with extremely sparse or high variance rewards, which are common in moneyline wagers where large payouts occur infrequently. This leads to unstable convergence or premature policy oscillations.

8.2 Implications for Real-World Esports Betting

Our findings suggest that integrating sophisticated market signals can improve an agent's profitability relative to naive heuristics.

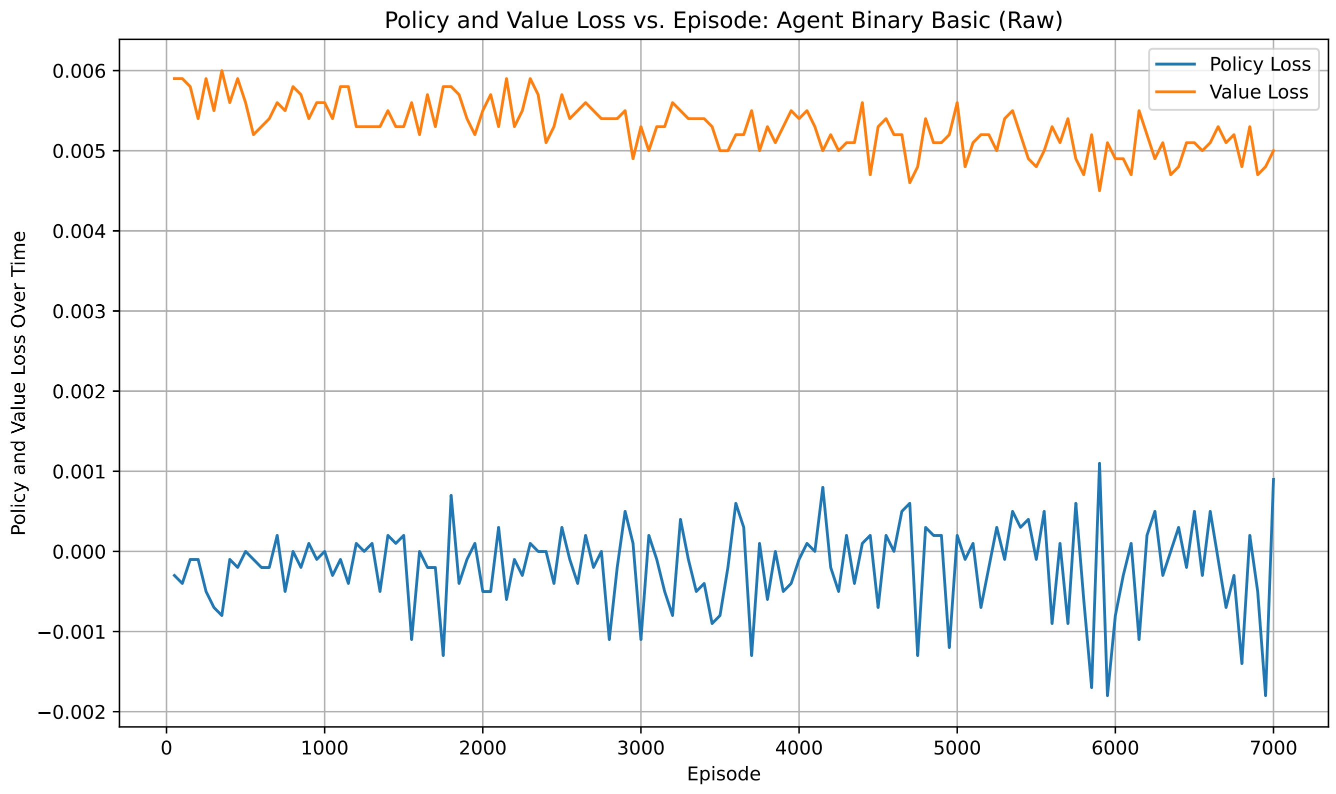

Figure 6: Cumulative reward during the training phase (all agents).

Despite this, several practical considerations must be addressed before real application. Real-money betting platforms impose transaction fees and limits on bet sizes. These are not modeled in our simulations. Ignoring these factors can lead to overestimation of returns and underestimation of risks.

Additionally, betting exchanges often adjust odds dynamically to balance their books, creating a feedback loop that our agent does not anticipate [5]. Finally, regulatory constraints vary by region and platform. As such, automated betting algorithms must adhere to strict compliance guidelines, which further limit the agent's ability to place rapid, high-frequency bets.

8.3 Ethical Considerations in Gambling Applications

Applying reinforcement learning to gambling raises important ethical questions. Automated agents could exploit gambling by enabling high-frequency wagering without human friction or emotional checks [7]. There is a risk that users might become over-confident on "black box" strategies, obscuring the inherent house edge and leading to financial harm.

Deploying such agents in live markets could unintentionally facilitate predatory practices, such as match-fixing or exploitation of retail bettors. It is therefore crucial to incorporate ethical guardrails. Agents should incorporate maximum bet limits, mandatory "cool-off" periods, and transparent performance reporting before any real-money implementation. Researchers and practitioners should collaborate with regulatory bodies and gambling organizations to ensure that reinforcement learning based betting systems enhance consumer welfare rather than undermine.

9 Conclusion

This study applies Proximal Policy Optimization (PPO) to moneyline betting in Counter-Strike 2 and shows that integrating economic and market-derived features significantly improves agent performance. Agents augmented with delta economy, return on investment, implied probabilities, and buy-type indicators converge more quickly, exhibit lower variance in policy and value losses, and achieve consistently higher cumulative rewards and bankroll growth compared to agents relying solely on raw game statistics. Crafted feature vectors also yield better-calibrated win probabilities and tighter return distributions, confirming the value of financial signals in esports wagering.

Practical limitations remain: PPO's on-policy updates demand fresh data in a dynamic betting market, and our simulation omits real-time odds updates, transaction fees, and overtime rounds. Addressing these gaps by incorporating dynamic odds, modeling fees, regulatory constraints, and extending to alternative learning methods can constitute clear future work. Ethical guardrails, such as bet limits and mandatory cooldowns, are essential to ensure responsible

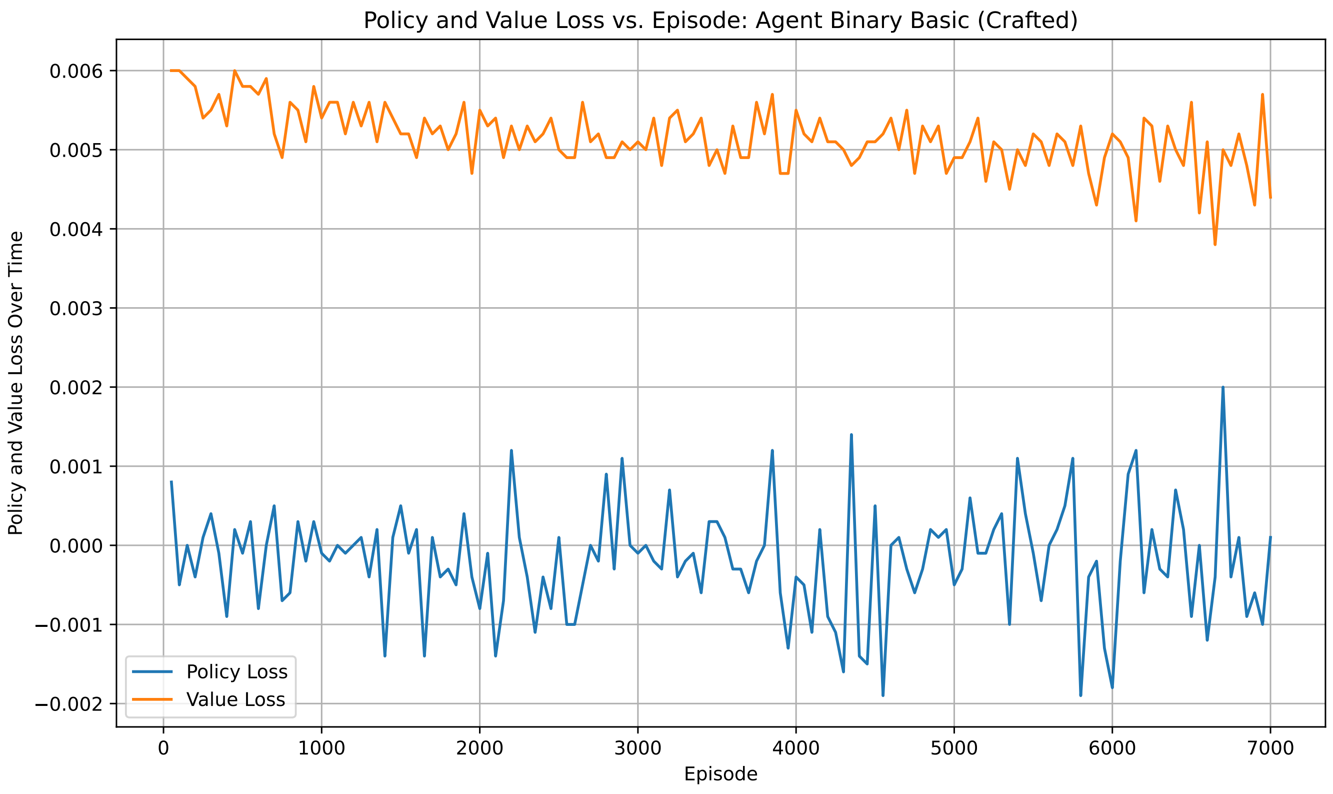

Figure 7: Cumulative reward during the training phase (all agents).

deployment. Overall, these results establish a foundation for applying reinforcement learning to esports betting while highlighting key considerations for live implementation.

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] DataCamp. 2023. One-Hot Encoding in Python: A Tutorial. https://www.datacamp.com/tutorial/one-hot-encoding-python-tutorial. Accessed: 2025-06-02.
[5] Brandon Jansen and Kyle Smith. 2020. Reinforcement Learning for Sports Betting: A Case Study on Soccer. *IEEE Transactions on Computational Intelligence and AI in Games* 12, 3 (2020), 203–215. doi:10.1109/TCIAIG.2020.2973132
[6] 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
[7] David King. 2019. Ethics of AI in Gambling: Risks and Regulations. *Journal of Gambling Studies* 35, 4 (2019), 1141–1158. doi:10.1007/s10899-019-09833-7
[8] 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

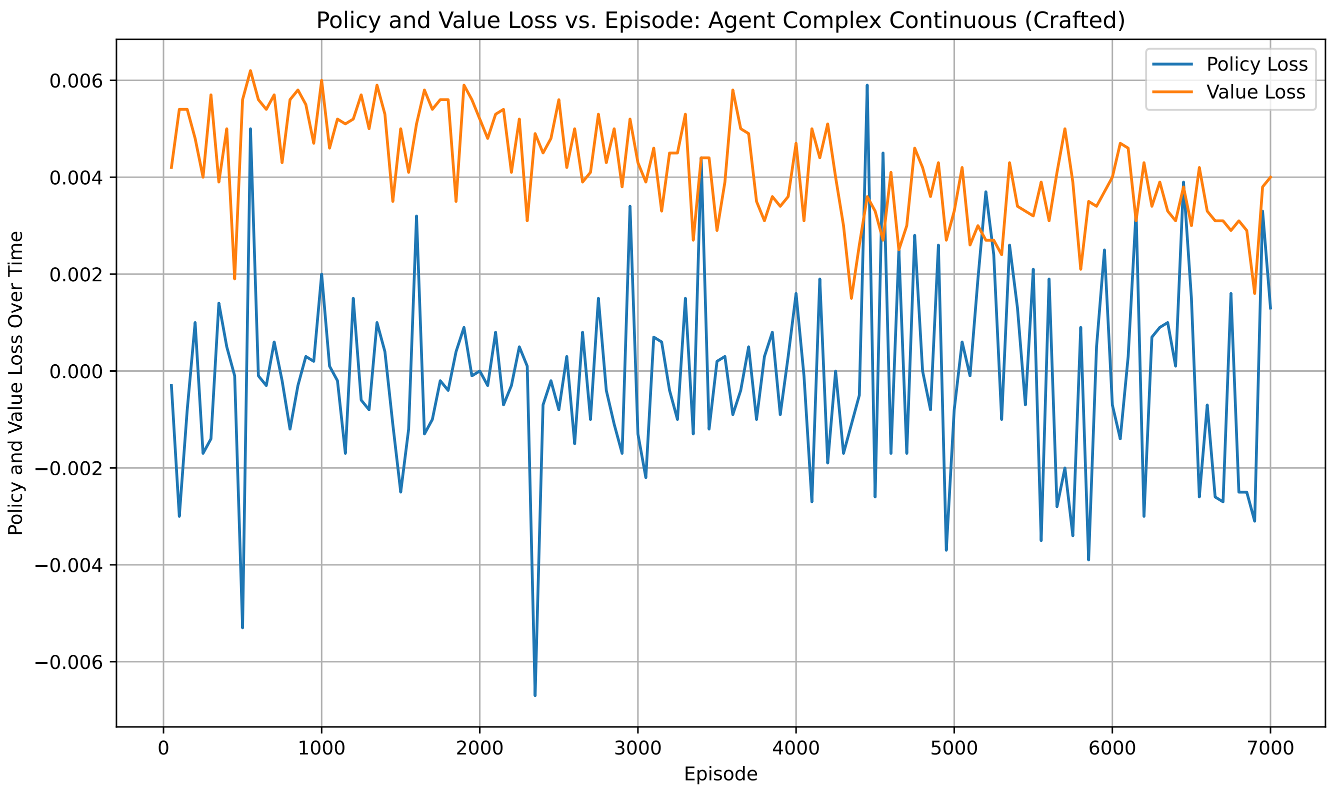

Figure 8: Cumulative reward during the training phase (all agents).

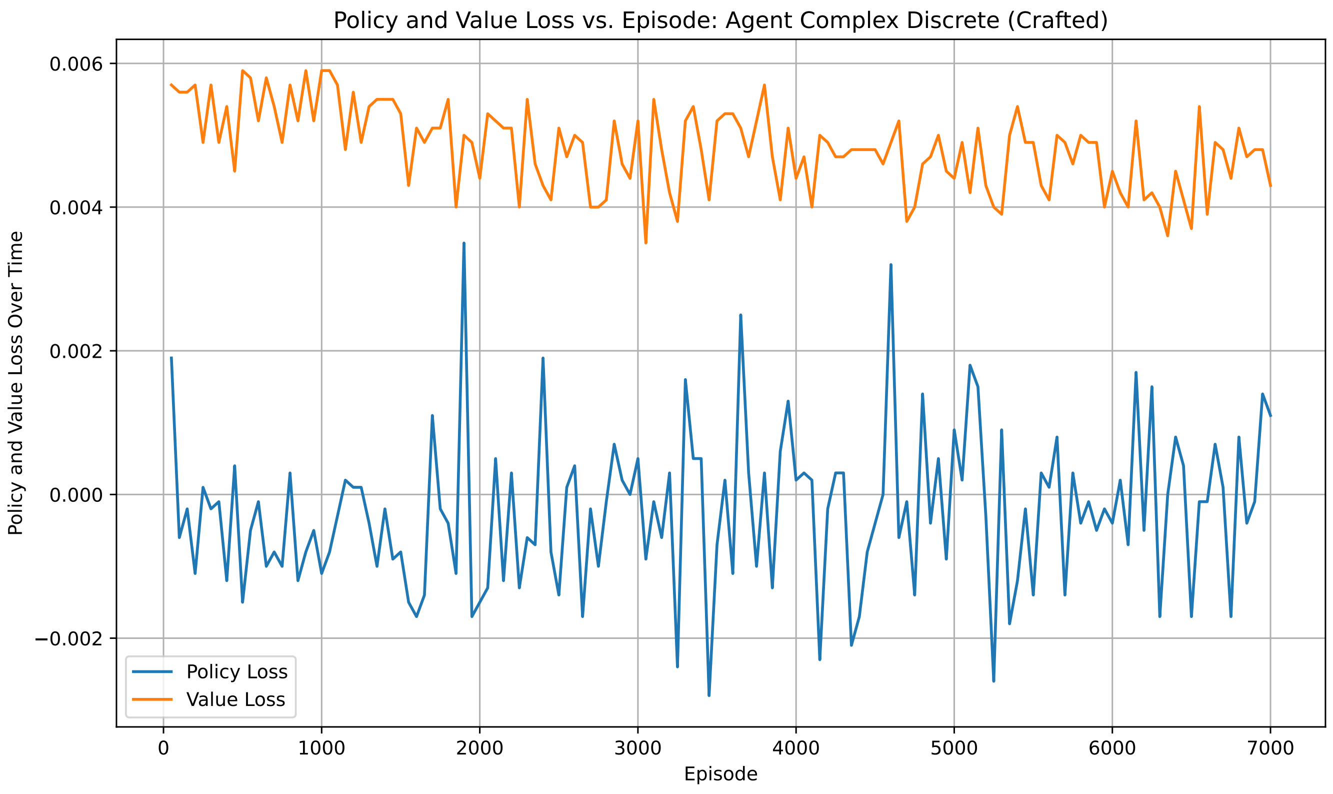

Figure 9: Cumulative reward during the training phase (all agents).

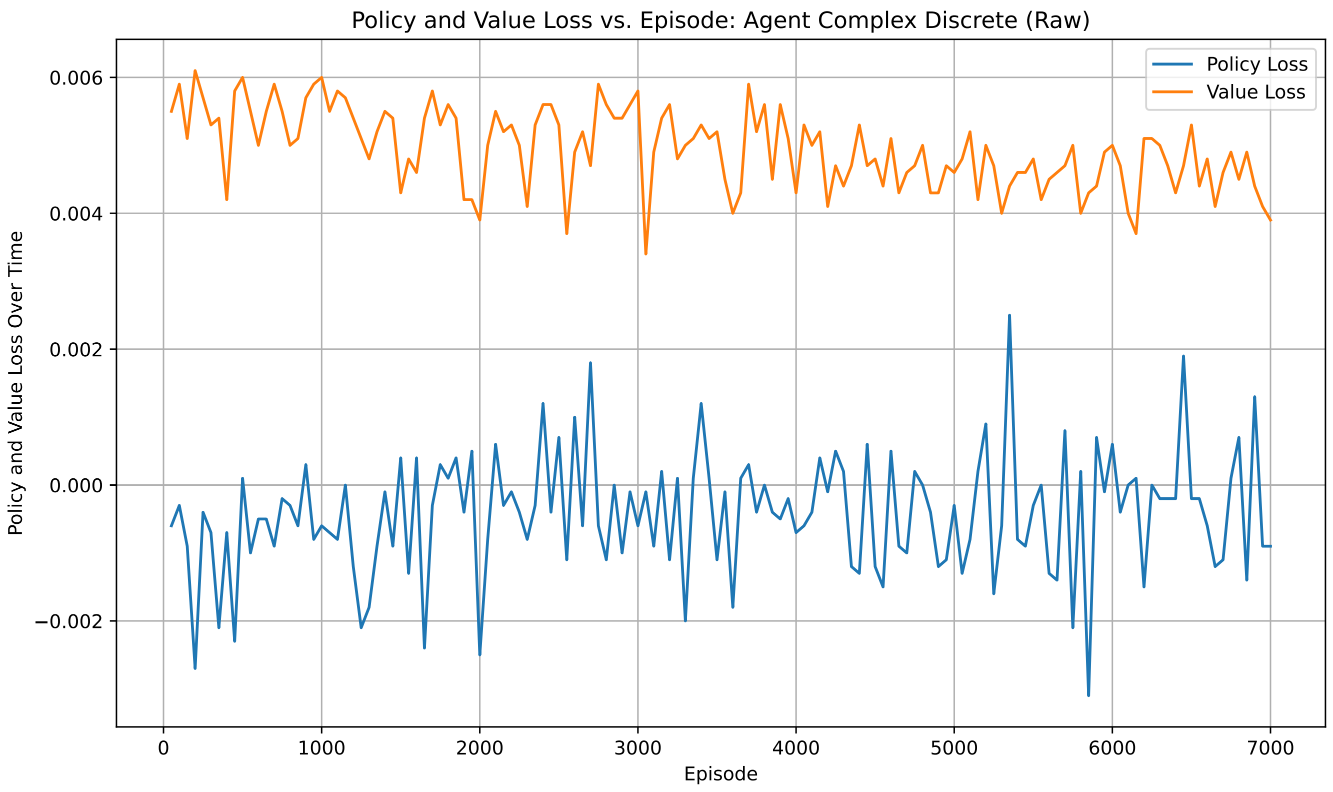

Figure 10: Cumulative reward during the training phase (all agents).

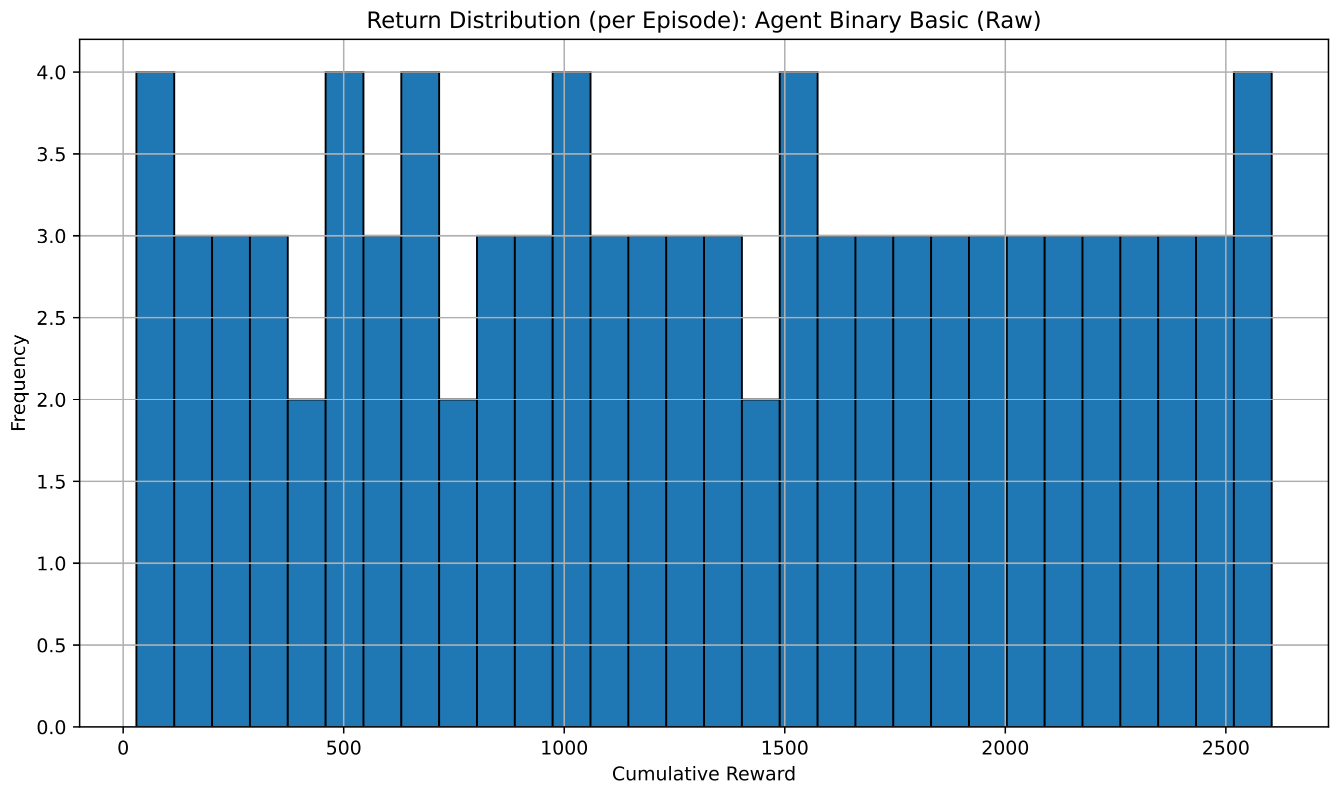

Figure 11: Distribution of episode returns for the Binary Basic (Raw) agent.

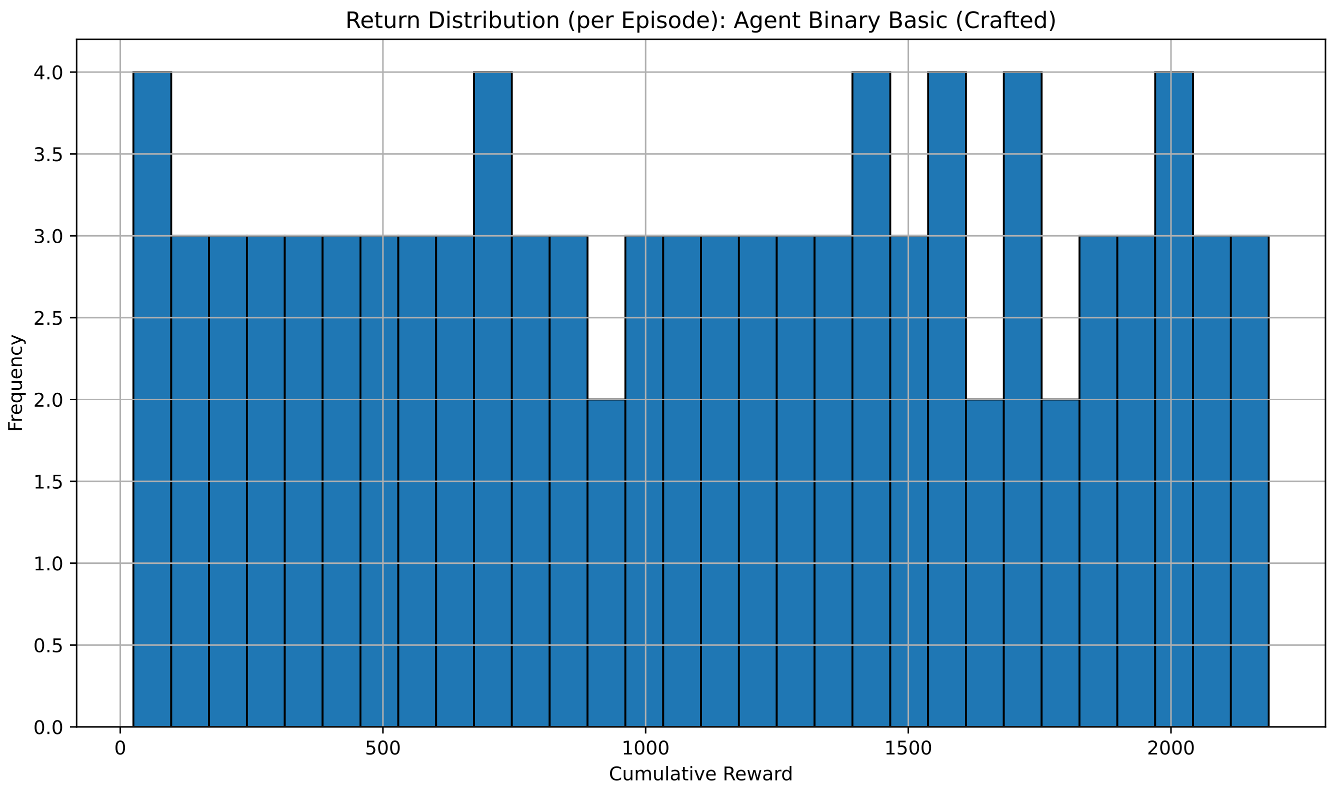

Figure 12: Distribution of episode returns for the Binary Basic (Crafted) agent.

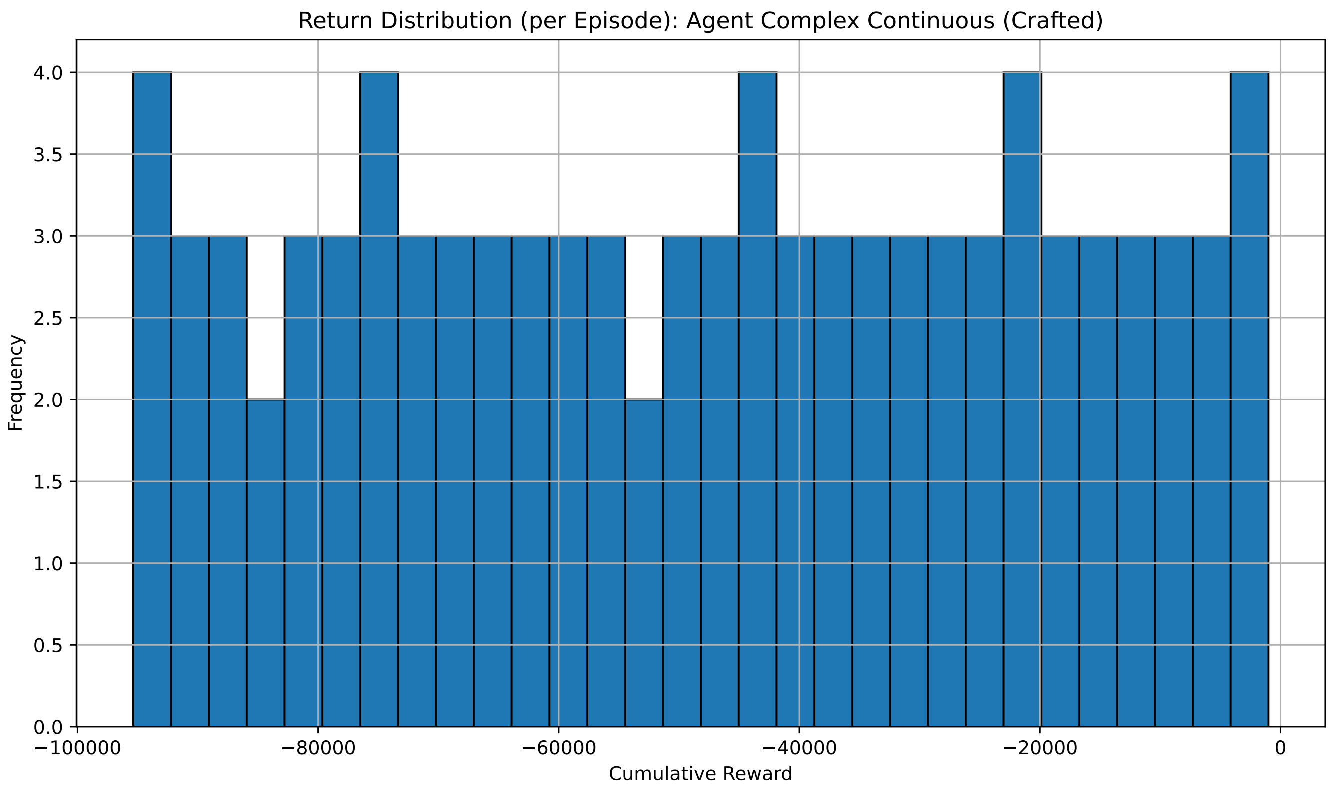

Figure 13: Distribution of episode returns for the Complex Continuous (Crafted) agent.

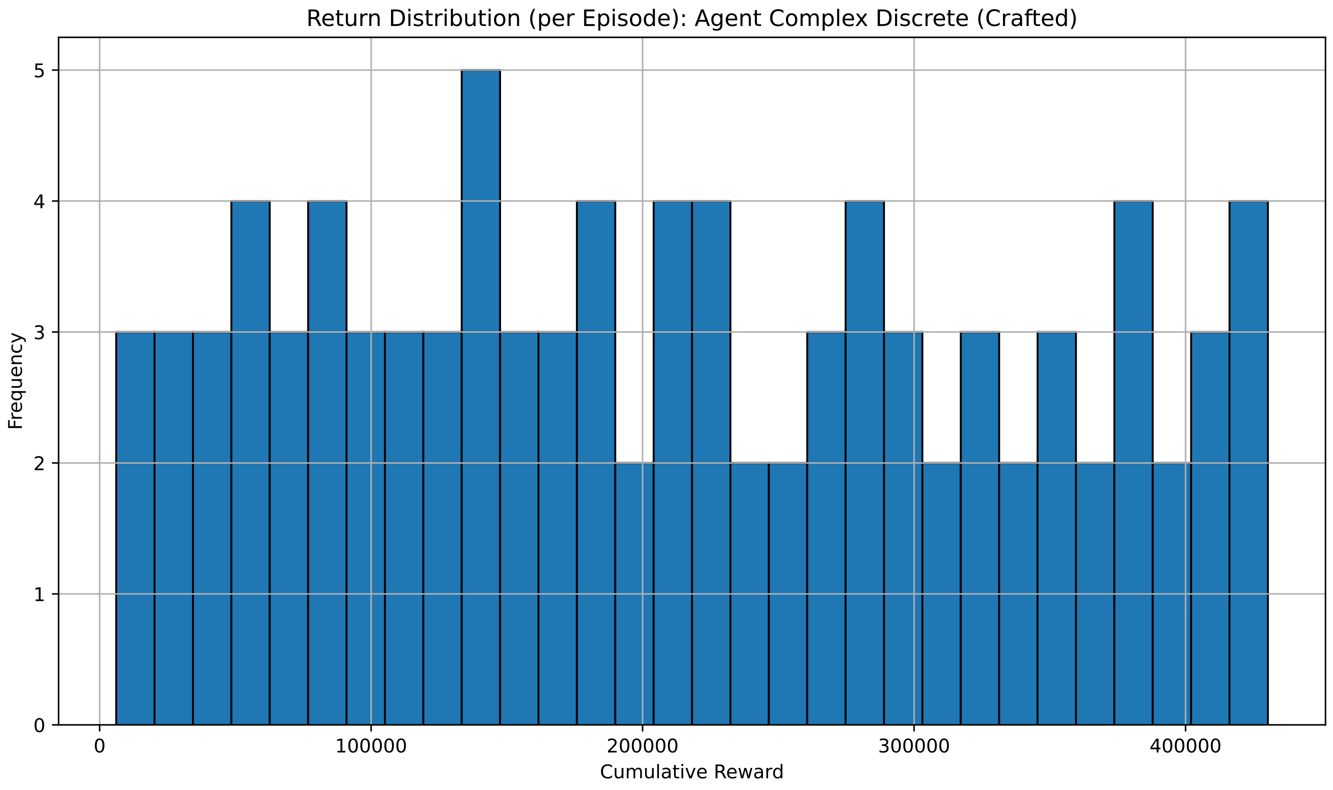

Figure 14: Distribution of episode returns for the Complex Discrete (Crafted) agent.

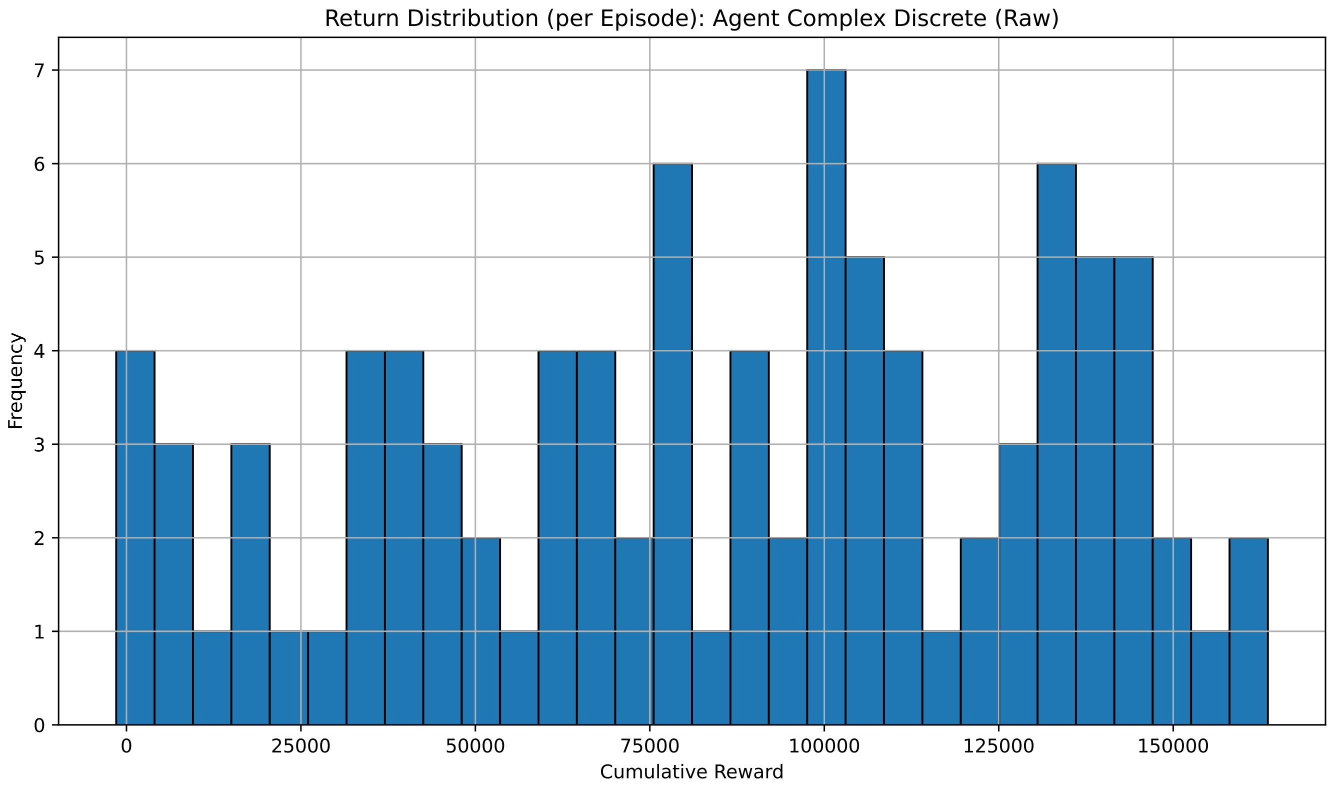

Figure 15: Distribution of episode returns for the Complex Discrete (Raw) agent.

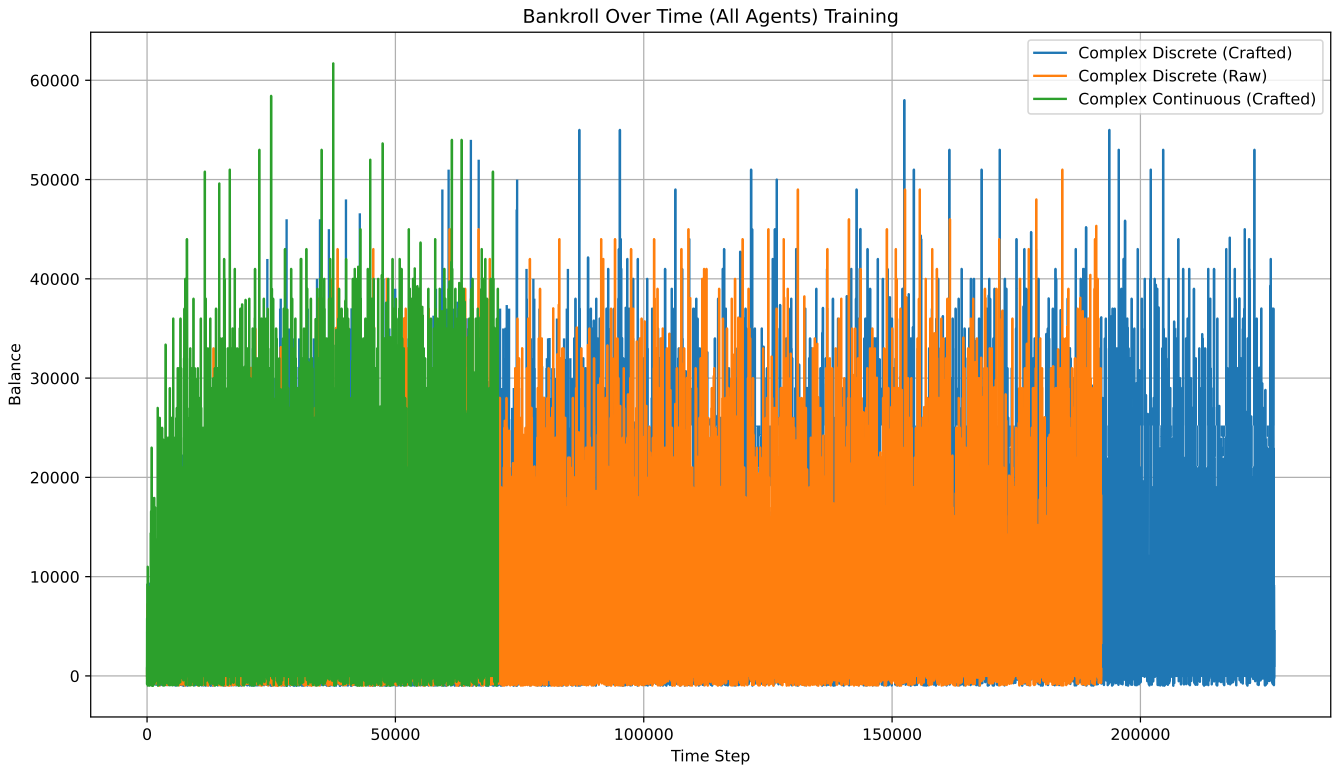

Figure 16: Bankroll growth over the training phase for all agents.

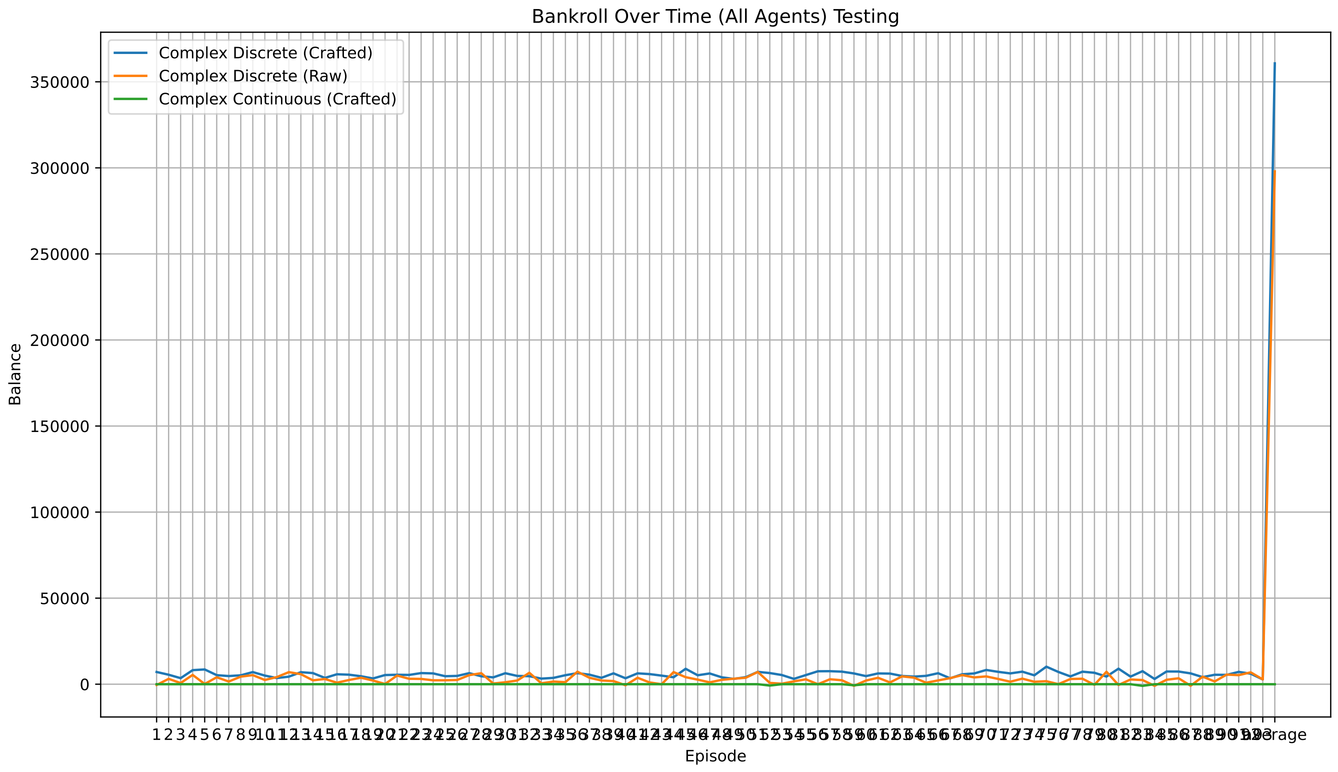

Figure 17: Bankroll growth over the testing phase for all agents.

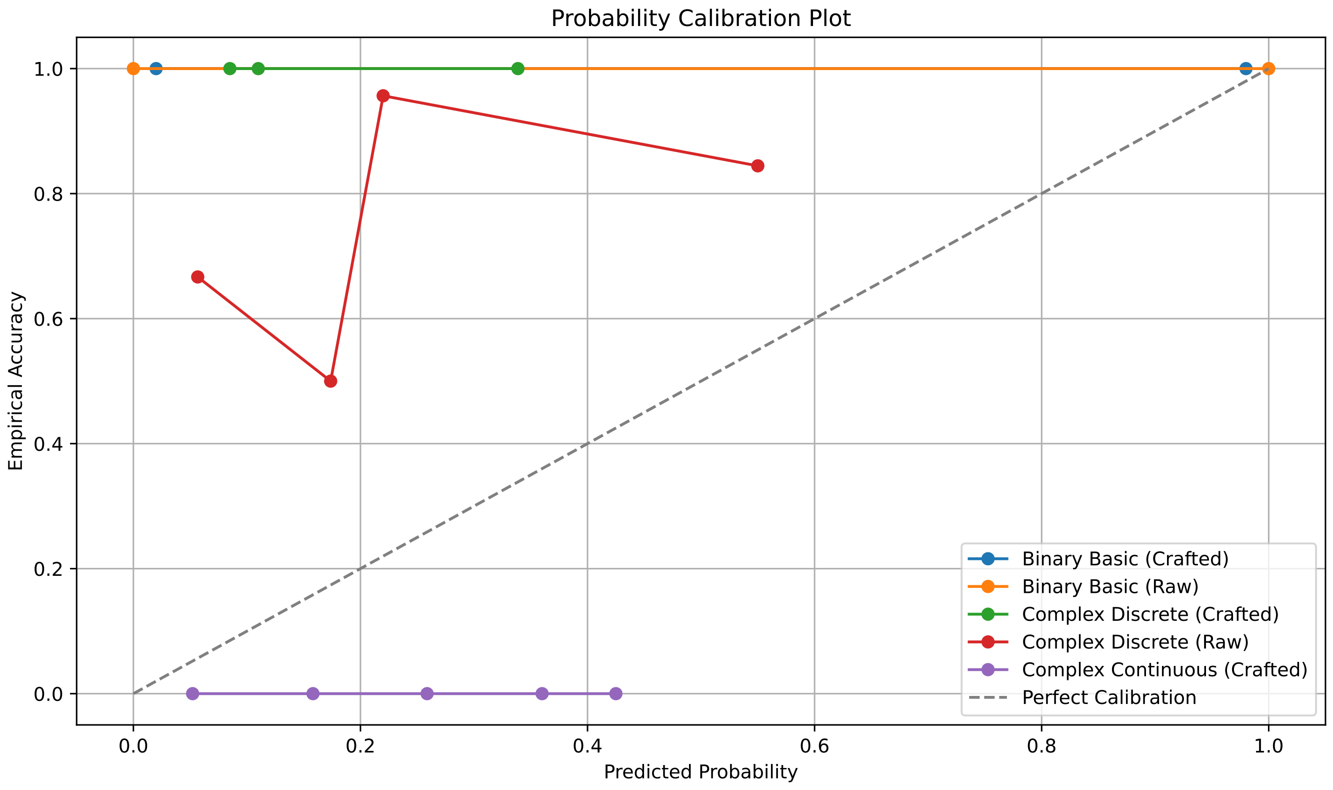
Probability Calibration Plot
Empirical Accuracy
Predicted Probability
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
Binary Basic (Crafted)
Binary Basic (Raw)
Complex Discrete (Crafted)
Complex Discrete (Raw)
Complex Continuous (Crafted)
Perfect Calibration

Figure 18