Learning to Bet: Reinforcement Learning for Live Wagering in Major League Baseball

Alico Redi* Sean Brindley Conlon* Shervin Jahanbakhsh* Stefan Uifalean*

Bocconi Students for Machine Learning, Bocconi University, Milan, Italy
{alico.redi sean.conlon shervin.jahanbakhsh stefan.uifalean}@studbocconi.it

July 16, 2025

Abstract

This project explores the use of reinforcement learning (RL) [6] to optimize live sports wagering strategies in Major League Baseball (MLB). We formulate the betting problem as a sequential decision process and develop an environment using historical play-by-play data combined with bookmaker odds. Due to the limited availability of historical odds, we train a supervised model to estimate missing prices, enabling large-scale RL training. We evaluate four RL algorithms—A2C, PPO, TRPO, and DSAC—each trained under four behavioral variants. Our best-performing model, PPO with reward shaping, achieves positive returns in simulation but fails to generalize fully to real bookmaker odds. We discuss limitations stemming from imperfect odds modeling, market inefficiencies, and structural bookmaker advantages. Our findings highlight the potential of RL in financial game environments and suggest future directions for improving realism and profitability.

1 Introduction

1.1 Problem Statement

This project aims to develop an algorithm for optimal live wagering during MLB games. Traditional machine learning approaches to sports betting typically assess wagers before the game starts. In contrast, we explore whether betting dynamically throughout the game yields better returns. This live setting enables game-theoretic strategies, such as in-game hedging, that are not possible in one-shot models. We evaluate various reinforcement learning algorithms [3] to address this question.

1.2 Motivation

Baseball is an ideal application for computational game solving for three reasons. First, baseball has an advanced sabermetrics community that provides many useful statistics. Second, baseball is a discretized game. In general, each game state can be thought of as a batter-pitcher matchup. The batter eventually produces an outcome that transitions the current game state to the next, with a new batter-pitcher matchup. Third, baseball is slow. Each batter-pitcher matchup takes on the order of a minute, providing ample time for an optimal wager strategy to be computed.

1.3 Environmental Setup

Live betting in baseball games can be formulated using a game-theoretic setup that naturally suits reinforcement learning algorithms:

Player actions

- **Bookmaker** – At each timestep, the bookmaker must set a price for each team. This price is a value greater than 1 that sets the multiplier for the payout to a winning wager.
- **Bettor** – At each timestep, the bettor can choose to make a wager within the limits of their bankroll at the price set by the bookmaker or choose to do nothing.

Player incentives and strategy equilibrium

- **Bookmaker** – Must set prices at an equilibrium to balance profitability and risk [1].

*Equal contribution, the ordering is alphabetical.

- **Bettor** – Must devise an optimal strategy to maximize expected utility against a best-response bookmaker.

1.4 Data Collection

Two data sources are needed: a game state vector and associated bookmaker prices.

Game States: Historical play-by-play records were parsed to create a dataset covering all regular season MLB games from 2000 to 2024. Features include basic state info (inning, outs, score), constant info (weather, park factor), batting and pitching stats aggregated over recent games.

Live Odds: Odds were scraped from 15 bookmakers every 5 minutes during active games in 2023–2024. These were matched to game states and averaged to reduce noise.

Challenge: Historical odds only available for 2023–2024.

Solution: Train a regression model on available data to predict odds for earlier seasons.

2 Related Work

We refer the reader to research on RL in financial trading [3], betting markets [1], and game theory applications [6]. This paper is among the first to integrate supervised odds estimation with RL training in a live sports context.

3 Methodology

3.1 Supervised Learning (Task 1)

To fill in missing bookmaker odds, we trained models to predict live odds from game states.

Models:

- **Ensemble Regressor:** Combines XGBoost, LightGBM, and Random Forest. Robust to heterogeneous input features.
- **LSTM:** Captures temporal trends across game states but underperforms due to slow baseball dynamics.

Performance: Ensemble MAE = 0.04, LSTM MAE = 0.12. The ensemble was used to generate odds for 2000–2022 games.

3.2 Reinforcement Learning (Task 2)

Algorithms:

- A2C – Baseline actor-critic method [6].
- PPO – Stable policy optimization using clipped updates [5].
- TRPO – Conservative updates via KL-constraint [4].
- DSAC – Models return distribution for risk-aware strategies [2].

4 Results

4.1 Supervised Learning

Ensemble model selected due to lower MAE. LSTM struggled with coarse 5-minute sampling and slow game transitions.

4.2 Reinforcement Learning

Table 1: Average net return (in units) for each RL model and training variant.

Variant	A2C	PPO	TRPO	DSAC
Ver1	-10.15	-7.55	-9.44	-4.65
Ver2	-6.29	-9.20	-9.14	-5.25
Ver3	-7.32	-7.29	-8.80	-4.97
Ver4	-5.07	**+0.41**	-7.89	-5.29

PPO with Ver4 was the only model to yield positive expected returns in evaluation, though not in test. DSAC was the most stable. TRPO and A2C underperformed, showing sensitivity to shaping.

5 Discussion

The results of our reinforcement learning experiments reveal several critical dynamics affecting agent performance, many of which stem from the imperfect nature of our simulation pipeline.

A particularly notable finding is that PPO with Ver4 training (win incentive + high exploration) achieved a positive average return on the evaluation set, but still incurred a loss on the test set. This discrepancy highlights a key limitation of our setup: the evaluation data comes from the same distribution used during training, where odds are generated by a supervised learning model trained on real bookmaker prices.

While this model performs well (mean absolute error of 0.04), it is not perfectly accurate. As a result, the RL agent is trained and validated in an environment that only approximates the real-world odds landscape. This mismatch means that strategies learned on the evaluation data may exploit imperfections or patterns in the predicted odds that do not generalize to the actual market, explaining the observed performance drop when tested on real odds data.

Another important factor is that the odds used in this environment are not *fair* odds. In an idealized market, the implied probabilities derived from bookmaker prices should sum to 1. However, in practice, the sum is greater than 1—reflecting the bookmaker's built-in profit margin (also known as the *vig*) [1]. This structural edge ensures that even an optimal bettor faces an uphill battle. In our simulation, this margin persists, meaning all RL agents are operating in a slightly adversarial environment by design.

Furthermore, it is worth noting that the odds used in our environment are not sourced from a single bookmaker. Instead, they represent the average odds across 30 different betting houses. This aggregation was necessary to stabilize noisy data and remove outliers, but it comes at the cost of underestimating the agent's profit potential. In real-world conditions, a bettor could selectively place wagers at the bookmaker offering the highest odds for a given outcome. Thus, our agents are inherently conservative in their profit estimates, and their true performance in a live multi-book environment could be materially better.

Taken together, these points demonstrate both the strengths and limitations of our current pipeline:

- It enables RL training on a wide variety of game states by extending historical data through a supervised odds model.
- It reveals promising behaviors from algorithms like PPO when incentivized with exploratory and reward shaping mechanisms.
- But it also underscores the fragility of performance when transferring from an approximated training environment to real-world test data.

Future work should aim to reduce this simulation gap by:

- Improving the accuracy of the odds prediction model,
- Allowing agents to access the best odds among available bookmakers, and
- Incorporating fair-odds benchmarks.

These enhancements could help build agents that not only perform well in simulation, but also translate their strategies effectively to real betting markets.

6 Conclusion

We investigated RL agents for live betting in MLB using historical data and inferred odds. We proposed a supervised learning pipeline to generate odds where missing, enabling extensive RL training. PPO with reward shaping emerged as the best performer in simulation but did not generalize fully. This underscores the importance of realism in training environments. RL has potential in betting applications, but success depends on careful modeling of market dynamics.

7 Acknowledgments

We thank Retrosheet.org for play-by-play data, and acknowledge the developers of open-source tools used in this project.

References

[1] Sharp Betting. How betting markets work: Odds, vig, and implied probabilities, 2021. https://sharpbetting.com/education/how-betting-markets-work.

[2] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[3] John Moody and Martin Saffell. Performance functions and reinforcement learning for trading systems and portfolios. *Journal of Forecasting*, 17(5-6):441–470, 1998.

[4] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

[5] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[6] Richard S Sutton and Andrew G Barto. Reinforcement learning: An introduction. *MIT Press*, 1998.

A Appendix

A.1 A. Retrosheet Notice

A.2 B. Project Repository

The full codebase and documentation for this project are available on GitHub at:
https://github.com/UifaleanStefan/MLBLiveBetting/tree/main. Data and trained models can be provided if requested by emailing the authors.

A.3 B. Game State Features

Batting Features: PA, K%, BB%, wOBA, wRAA.
Pitching Features: TBF, K%, BB%, GO/TBF, FIP.

A.4 C. RL Model Descriptions

A2C: Synchronous actor-critic with advantage-based updates. Reduces variance but sensitive to sparse rewards. **PPO:** Uses clipped surrogate objective to limit policy changes. Stable and efficient. **DSAC:** Learns return distributions for improved robustness and risk sensitivity. **TRPO:** Optimizes policy under KL-divergence constraint. Stable but computationally demanding.