Optimal Trade Execution via Deep Reinforcement Learning

Sankalp Yadav

June 26, 2025

Abstract

The execution of large orders in financial markets is a critical task for institutional investors, where naive strategies can lead to significant costs from market impact. This paper proposes a framework for developing an intelligent agent that learns an optimal execution policy using Deep Reinforcement Learning (DRL). I frame the optimal execution problem as a finite-horizon Markov Decision Process (MDP). The agent's goal is to minimize a combination of implementation shortfall and volatility risk by adaptively choosing how to parcel out a large parent order over a series of discrete time steps. I will implement a state-of-the-art DRL algorithm, Proximal Policy Optimization (PPO), to train the agent in a simulated market environment built upon high-frequency, publicly available limit order book data. The performance of the trained DRL agent will be rigorously evaluated against standard industry benchmarks, namely Time-Weighted Average Price (TWAP) and Volume-Weighted Average Price (VWAP) strategies. This research aims to demonstrate that a data-driven, model-free DRL approach can discover sophisticated execution strategies that outperform static and semi-static benchmarks in realistic market conditions.

1 Introduction

The task of liquidating a large financial position, known as trade execution, is a cornerstone of quantitative finance. A naive approach, such as placing the entire large order on the market at once, would create a significant price shock, leading to substantial execution costs. This cost, often termed **market impact**, is the adverse price movement caused by the trading activity itself. Consequently, practitioners break down large "parent" orders into a sequence of smaller "child" orders, executing them over a period of time to mitigate this impact.

The central challenge is to find the optimal schedule of these child orders. A slow execution minimizes market impact but exposes the trader to **volatility risk**—the risk that the market price moves unfavorably during the extended execution horizon. This creates a fundamental trade-off, which has been the subject of extensive research.

The seminal work by **Almgren and Chriss (2001)** provided a foundational analytical framework for this problem, balancing market impact costs against timing risk using dynamic programming. However, their models rely on strong assumptions about market dynamics and impact functions. More recently, the proliferation of high-frequency data and advancements in machine learning have opened new avenues for tackling this problem.

Reinforcement Learning (RL) is a particularly suitable paradigm. It allows an agent to learn a policy through direct interaction with an environment, without needing an explicit model of market dynamics. Recent works have shown the promise of applying Deep Reinforcement Learning (DRL), which uses deep neural networks as function approximators, to complex financial tasks. For instance, research has demonstrated DRL's effectiveness in portfolio optimization and market making. This project builds upon this frontier by designing, training, and evaluating a DRL agent specifically for the optimal execution problem.

My contribution is to provide a complete, practical blueprint for developing a high-performance DRL execution agent using publicly available data. I will meticulously define the problem as a Markov Decision Process (MDP), build a realistic simulation environment that accounts for the agent's own market impact, and benchmark my agent's learned policy against widely used industry strategies.

2 Data Selection and Justification

A core requirement for this research is a high-fidelity market simulator capable of modeling the agent's impact on a Limit Order Book (LOB). This necessitates access to high-frequency, full-depth LOB data. The choice of the underlying market—traditional equities versus digital assets—is therefore a critical first step determined largely by data accessibility.

For an individual researcher, acquiring the necessary data for traditional equities is prohibitive. Institutional-grade LOB data feeds for major stock exchanges (e.g., NASDAQ TotalView, NYSE TAQ) are extremely expensive and complex. Furthermore, the equities market is highly fragmented, meaning a single stock trades across numerous lit exchanges and dark pools simultaneously. A simulation based on a single venue would therefore be an inaccurate and incomplete representation of the true market.

In contrast, the cryptocurrency market offers a pragmatic and powerful alternative.

1. **Data Accessibility**: Most major cryptocurrency exchanges (e.g., Binance, Coinbase Pro, Kraken) provide free, public, and well-documented APIs for accessing both real-time and historical high-frequency LOB and trade data. This democratizes access to the granular data needed for this project.

2. **Market Simplicity**: While a single crypto asset trades on multiple exchanges, each exchange acts as a self-contained market. The LOB for BTC/USDT on Binance represents the complete market for that specific venue. This allows for the construction of a highly realistic, non-fragmented, single-venue simulator, making the research problem more tractable and the results more valid within that environment.

3. **Data Richness**: Crypto markets operate 24/7, providing a continuous, uninterrupted data stream ideal for training machine learning models without the complexity of handling market open/close periods or overnight gaps.

Therefore, for reasons of accessibility, tractability, and data integrity, I will use high-frequency LOB data from a major cryptocurrency exchange. The principles of market microstructure, impact modeling, and reinforcement learning are universal, and the skills developed on this data are directly transferable to any electronic market, including traditional finance.

3 The Optimal Execution Problem as an MDP

To apply RL, I formally define the optimal execution problem as a finite-horizon Markov Decision Process (MDP), characterized by the tuple $(\mathcal{S}, \mathcal{A}, P, \mathcal{R}, \gamma)$.

3.1 State Space ($\mathcal{S}$)

The state $s_t \in \mathcal{S}$ at time step t must provide the agent with a comprehensive snapshot of the market and its progress on the execution task. I define the state vector as a concatenation of the following normalized features:

- **Inventory Percentage**: I_t/I_0, where I_t is the remaining inventory to be executed and I_0 is the initial parent order size. This tells the agent how much work is left.
- **Time Percentage**: t/T, where T is the total number of time steps in the execution horizon. This provides a sense of urgency.
- **Limit Order Book (LOB) State**: To capture market liquidity and short-term price signals, I include:
  - Bid-Ask Spread: $(P_t^{ask} - P_t^{bid})/P_t^{mid}$.
  - LOB Imbalance: $(V_t^{bid} - V_t^{ask})/(V_t^{bid} + V_t^{ask})$, where V_t^{bid} and V_t^{ask} are the volumes available at the best bid and ask.
  - Prices and volumes at the first k levels of the LOB (e.g., $k = 5$), normalized by the mid-price and total volume.
- **Market Activity**: A short history (e.g., last 10 steps) of mid-price returns and traded volumes to capture momentum and volatility.

3.2 Action Space ($\mathcal{A}$)

At each time step t, the agent must decide what fraction of the *remaining* inventory to execute. I define the action $a_t \in \mathcal{A}$ as a continuous value:

$$a_t \in [0, 1] \tag{1}$$

The quantity to be sold in time step t is then $q_t = a_t \times I_t$. An action $a_t = 1$ means liquidating the entire remaining inventory at once.

3.3 Reward Function ($\mathcal{R}$)

The reward function is designed to incentivize minimizing execution costs. The agent receives a reward r_t at the end of each step t. A common and effective choice is to reward the agent based on the value it captures relative to the price at the beginning of the step. For a sell order, this is:

$$r_t = q_t \times (P_t^{exec} - P_t^{mid}) \tag{2}$$

where q_t is the quantity sold, P_t^{exec} is the average execution price for that quantity, and P_t^{mid} is the mid-price just before the trade. This formulation directly penalizes negative slippage (market impact). A large terminal penalty can be added if $I_T > 0$ to ensure the agent liquidates its entire inventory by the deadline T.

4 Methodology

4.1 Simulation Environment

A realistic simulation environment is the most critical component of this project. It must replay historical LOB data and model the agent's impact on that data.

1. **Data Source**: As established, I will use high-frequency LOB data for a liquid cryptocurrency pair (e.g., BTC/USD) from an exchange like Coinbase or Binance. This data, acquired via public APIs or platforms like Kaggle, will contain time-stamped LOB snapshots or updates.

2. **Market Replay**: The simulator will step through the historical data in discrete time intervals (e.g., every 10 seconds), presenting the agent with the market state at each step.

3. **Market Impact Model**: When the agent places a child order of size q_t, it will "eat" through the LOB. The average execution price P_t^{exec} is calculated based on the volume-weighted average of the price levels consumed by the order. I will also model a **permanent market impact**, where the agent's trade slightly shifts the mid-price for subsequent steps, based on a simple linear model of the traded volume.

4.2 Deep Reinforcement Learning Algorithm: PPO

I will use the **Proximal Policy Optimization (PPO)** algorithm, a state-of-the-art policy gradient method known for its stability and performance. PPO operates in an Actor-Critic framework.

- **Actor**: A neural network that takes the state s_t as input and outputs the parameters of a probability distribution over the action space (e.g., the mean and standard deviation of a Beta or Gaussian distribution, from which a_t is sampled).

- **Critic**: A second neural network that takes the state s_t as input and outputs an estimate of the value of that state, $V(s_t)$. This is used to reduce the variance of the policy gradient updates.

PPO's key innovation is its clipped surrogate objective function, which discourages the policy from changing too drastically at each update step, leading to more stable training. I will use a standard implementation of PPO from a reliable library like Stable Baselines3 or Tianshou.

4.3 Benchmark Strategies

To evaluate the DRL agent, I will implement two universally recognized benchmarks:

- **Time-Weighted Average Price (TWAP)**: This strategy is static and aims to execute an equal fraction of the total order in each time interval.

$$q_t^{TWAP} = \frac{I_0}{T} \tag{3}$$

- **Volume-Weighted Average Price (VWAP)**: This semi-static strategy aims to participate in proportion to the historical volume distribution. I will pre-calculate a historical intraday volume profile and have the VWAP agent execute a fraction of the parent order in each interval that is proportional to this profile.

5 Experimental Design

1. **Dataset**: I will acquire LOB data for a liquid asset over a period of several months. I will split this chronologically into a **training set** (e.g., first 70% of the data) and a **testing set** (final 30%). The agent will never see the test data during training.
2. **Training Phase**: The PPO agent will be trained on the training set for a large number of episodes. Each episode consists of a randomly selected starting point in the training data and a fixed execution task (e.g., "sell 100 BTC over 30 minutes").
3. **Evaluation Phase**: After training, the agent's policy is frozen. It is then evaluated on thousands of execution tasks drawn from the unseen test set. The same tasks will be executed by the TWAP and VWAP benchmarks.
4. **Performance Metrics**: I will compare the strategies based on the following metrics, averaged over all test episodes:
    - **Implementation Shortfall (IS)**: This is the primary cost metric. It is the difference between the value of the order at the decision time (e.g., $I_0 \times P_0^{mid}$) and the final cash received from the liquidation. Lower IS is better.
    - **Slippage**: The average execution price relative to the initial mid-price ($P_{avg}^{exec} - P_0^{mid}$).
    - **Risk**: The standard deviation of the Implementation Shortfall across all test episodes. A lower standard deviation indicates a more consistent and reliable strategy.

6 Expected Results and Conclusion

I expect to find that the DRL agent learns a non-trivial, dynamic execution policy. I hypothesize that the agent will learn to:

- Trade more aggressively when the LOB is deep and liquid (low spread, high volume).
- Trade more passively when the market is thin to avoid high impact.
- Potentially exploit short-term momentum signals present in the market activity features.

I anticipate that this adaptive behavior will allow the DRL agent to achieve a statistically significant lower average Implementation Shortfall compared to the static TWAP and semi-static VWAP benchmarks, while maintaining a comparable level of risk.

A successful outcome will provide strong evidence for the practical applicability of DRL in creating next-generation execution algorithms that can adapt to real-time market conditions, thereby delivering superior performance for institutional trading.

7 Future Work

This research can be extended in several directions:

- Incorporating additional data streams into the state space, such as news sentiment from NLP models.
- Designing a multi-agent framework where agents compete for liquidity.
- Applying the framework to a portfolio execution problem, where correlations between assets must be managed.

References

[1] R. Almgren and N. Chriss, "Optimal execution of portfolio transactions," *Journal of Risk*, vol. 3, pp. 5-40, 2001.

[2] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," arXiv preprint arXiv:1707.06347, 2017.

[3] B. Liu, H. Yang, and Q. Wang, "FinRL: A Deep Reinforcement Learning Library for Automated Stock Trading in Quantitative Finance," arXiv preprint arXiv:2011.09607, 2020.