Optimal Liquidation using Proximal Policy Optimisation in a Non-Parametric K-Nearest-Neighbour Limit Order Book simulator

Louie Spinks

September 15, 2025

Abstract

Optimal execution of large financial orders involves a fundamental trade-off between market impact and timing risk. Classical analytical frameworks, such as the Almgren-Chriss solutions, rely on simplifying and restrictive assumptions about the underlying market dynamics, limiting their effectiveness in real markets. This dissertation develops a data-driven framework to address these limitations. First, a non-parametric Limit Order Book simulator is constructed using the statistical learning K-Nearest Neighbour (K-NN) resampling method, which implicitly models market dynamics by using historically realised market state evolutions. The simulator is empirically validated to reproduce key statistical properties, including the non-linear square-root scaling of market impact (scaling exponent $\alpha = 0.493$). Within this simulation environment, a Deep Reinforcement Learning (DRL) agent is trained using the Proximal Policy Optimisation algorithm, for the task of optimal trade execution. The agent's unique formulation includes an explicit consideration for risk in its reward function. When tasked with liquidating a large position in MSFT stock, the trained agent significantly outperforms a standard TWAP benchmark strategy, achieving a 62% reduction in mean implementation shortfall and a 77.6% reduction in its standard deviation. Our results validate the novel combination of non-parametric K-NN simulation with a risk-aware DRL agent to create adaptive, high-performance trade execution strategies.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 The Cost of Institutional Trading

In modern financial markets, the effective execution of portfolio transactions is a highly relevant and a critical determinant of investment performance. This problem is particularly acute for institutional investors, who regularly face the challenge of executing large transactions that can, if handled poorly, incur significant costs–eroding any potential returns from the trade. The mechanics of this challenge are rooted in the structure and dynamics of today's electronic exchanges, which are predominantly organised as centralised Limit Order Books (LOBs), in which market participants publicly quote prices at which they are willing to buy/sell some quantity of a given asset.

The problem institutional traders face when executing large positions can be decomposed into two fundamental and conflicting sources of cost:

- **Market Impact**: The cost of consuming liquidity, which is composed of two effects. First, a *temporary impact*, when executing a large market order, the trader is exposed to progressively worse prices as their order depletes the liquidity of the posted bid/ask quotes, temporarily shifting the asset price away from the equilibrium due to the mechanically imposed shift in supply and demand. Second, a *permanent impact*, an adverse price shift resulting from the market inferring information from

the trade, which can persist for the duration of the trading horizon.

- **Timing Risk**: The cost of price uncertainty over time. To mitigate market impact, the trader may divide the large "parent" order into smaller "child" orders to be executed over a given horizon. This, however, exposes the unexecuted portion of the portfolio to the natural volatility and drift of the asset's price.

Concretely, the problem of optimal trade execution may then be defined as finding the trading schedule that transforms the initial portfolio to a specified final state within a given horizon while minimising cost or more specifically the *Implementation Shortfall*– the difference between the realised trade revenue and the theoretical portfolio value at the start of the trading horizon. Furthermore, in defining this notion of optimality, a rational, risk-averse trader should seek a trading schedule that not only minimises the expected implementation shortfall but also its variance–this can often require an adaptive approach.

1.2 Classical Approaches to Trade Execution

In response to the execution costs outlined previously, several common benchmark strategies have been developed and adopted by industry. Such strategies are often static; their trading schedule is determined *a priori* and is not adapted according to intra-day market conditions. The most fundamental of these, the *Time-Weighted Average Price* (TWAP) strategy, involves dividing the parent order into equally-sized child orders that are executed at regular intervals over the trading horizon. While this method can reduce the market impact of a single large order, its static nature leaves it heavily exposed to timing risk.

The seminal work from Almgren & Chriss [1] provided the foundational framework for optimal trade execution. Their approach constructs an "efficient frontier" of static trading strategies, where each element represents the strategy with the lowest expected shortfall and variance for each given level of a trader's risk aversion. There are however

some critical limitations to this construction. In particular, their formulation requires the assumption of no serial correlation in the asset's price, that is, the price dynamics are assumed to be a random arithmetic walk with no temporal correlation and no price drift. Furthermore, their framework requires a pre-specified parametric representation of market impact, and although not required, in their explicit construction they assume a linear market impact model–contradictory to the well documented "stylised fact" of a concave market impact [2].

The limitations outlined above provide clear motivation for a dynamic approach to optimal trade execution.

1.3 Reinforcement Learning & LOB Simulation

The restrictions imposed by analytical models prompt a necessary shift towards a dynamic, data-driven approach. Reinforcement Learning (RL) provides a natural framework for this challenge, in which the problem may be formulated as a sequential decision-making process. This shift, however, will require a reliable method of LOB simulation.

A thorough survey of LOB simulation methods can be found in the recent review by Jain et al.[3]. Many of the established approaches, such as those based on Point Processes or Stochastic Differential Equations, are inherently parametric, often relying on strong assumptions about the underlying data-generating process. This reliance on explicit assumptions risks misrepresenting the true market dynamics.

To circumvent these limitations, this dissertation adopts a non-parametric approach to LOB simulation. We implement the K-Nearest-Neighbour (K-NN) resampling method recently proposed by Giegrich et al.[4], which operates on the principle that similar LOB states in a metric sense lead to similar state transitions. This idea allows for the construction of many plausible market trajectories directly driven from historical data, creating a simulator that captures complex dynamics without imposing strong modelling biases.

Within this simulated LOB environment, a RL agent is tasked with learning a policy– a mapping from observed market states to optimal trading actions. This learned policy is inherently adaptive, allowing the agent to dynamically react to evolving market conditions. For this task, we employ an Actor-Critic algorithm, using the Proximal Policy Optimisation (PPO) procedure developed by Schulman et al.[5]–a robust policy gradient method known for its sample efficiency.

1.4 Overview

The remainder of this dissertation is structured to systematically develop, implement and validate a dynamic approach to optimal trade execution using Deep Reinforcement Learning (DRL), within a non-parametric LOB simulation environment.

In Chapter 2 we provide the theoretical foundation for this work, including a review of existing approaches to optimal execution and LOB simulation. In Chapter 3, we outline our methodological approach to LOB simulation and the construction of our DRL trading agent, both of which are novel formulations. Chapter 4 presents the evaluation of our constructed framework, in which we validate the statistical fidelity of our trading simulator and provide a comparative performance analysis of our trained optimal execution agent against a standard industry TWAP benchmark. Finally, Chapter 5 summarises and contextualise our work.

Chapter 2

Literature Review

2.1 Limit Order Books

Limit Order Books serve as the main centralised mechanism for facilitating market transactions, they maintain a real-time record of all outstanding commitments to buy or sell a quantity of a given asset at specified prices, effectively capturing the instantaneous state of supply and demand. The book is composed of two sides; the *bid* side, containing orders to buy, and the *ask* side, containing orders to sell.

Market participants may interact with the LOB through two main order types, which reflect alternate approaches to execution:

- **Limit Orders**: These are passive orders that provide liquidity to the market, in which the trader can specify the volume (No. of shares) and price at which they wish to buy/sell. Limit orders are arranged in the book according to a price-time priority rule, that is, the best priced bid/ask orders are placed to the "top" of the book (the highest bid and the lowest ask) which are then first to be executed against incoming orders. Furthermore, multiple orders submitted at the same price level are executed according to their time of submission.[1]

[1]This is the most common matching sequence used in equities markets, though other matching procedures do exist.

- **Market Orders**: These are aggressive orders that consume liquidity from the book, submitted by traders seeking immediate execution. The order specifies a quantity but not a price, and is executed against the state of the book upon receipt. The order depletes the liquidity of the limit orders at the top of the book, sequentially "walking the book" until the order is fulfilled. Buy market orders are executed against the ask side and conversely for sell market orders.

An instantaneous snapshot of the LOB can be found illustrated in 2.1, with limit orders populating discrete, tick-separated price levels. With the top of the book centred about the *mid-price*–a key proxy for the asset's current share value. We define price levels with non-zero limit order volume to be "active".

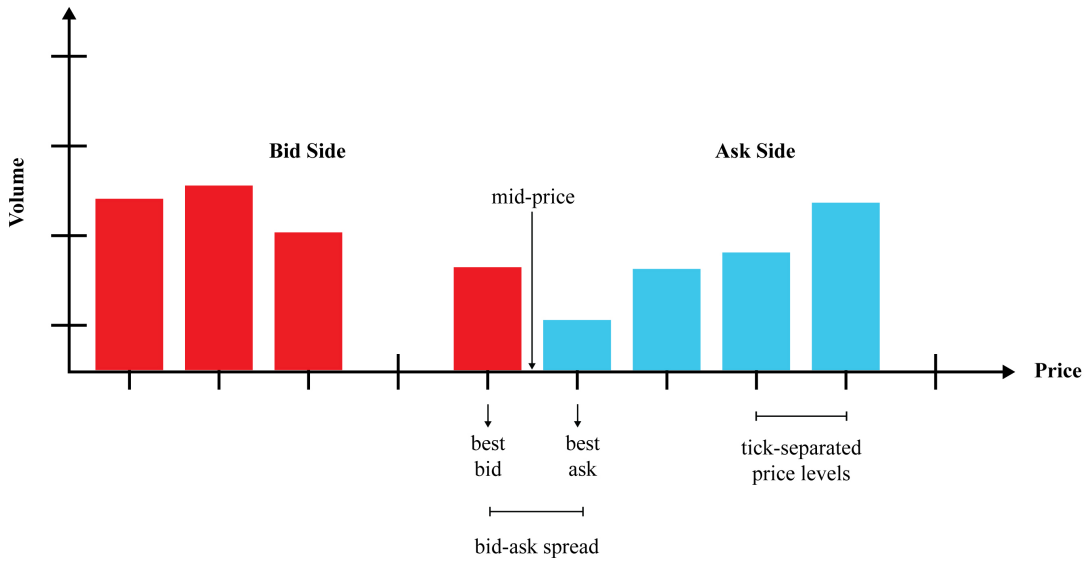

Figure 2.1: Graphical representation of the Limit Order Book, showing the distribution of limit orders occupying discrete tick-separated ($0.01) price levels.

Having established this structure, we can now formally introduce some key metrics and establish the notation that will be used throughout this dissertation:

Metric	Definition
$p^a_{t,i}$	The i^{th} discrete active price level on the ask side, where $p^a_{t,1}$ is the best ask price.
$p^b_{t,i}$	The i^{th} discrete active price level on the bid side, where $p^b_{t,1}$ is the best bid price.
p^*_t	The mid-price, given by: $p^*_t := \frac{p^a_{t,1}+p^b_{t,1}}{2}$.
w_t	The bid-ask spread, given by: $w_t := p^a_{t,1} - p^b_{t,1}$.
$V^a_{t,i}$	The volume available at the i^{th} active price level on the ask side.
$V^b_{t,i}$	The volume available at the i^{th} active price level on the bid side.

Table 2.1: LOB metric definitions.

2.2 Optimal Trade Execution

As outlined in Chapter 1, the optimal execution problem seeks the trading strategy that minimises the expected costs associated with executing a large position. In this dissertation we consider the sell-side of optimal execution, through market sell orders, though the problem is completely analogous for the buy-side. Specifically, we consider an initial holding of X shares, that is to be fully liquidated over the given trading horizon $[0, T]$.

2.2.1 General Problem Formulation in Discrete Time

To formalise the problem, we adopt a discrete-time framework common in both classical and computational RL approaches. We divide the trading horizon $[0, T]$ into N equal intervals of length $\Delta t = T/N$, creating N discrete time-steps, $t_k = k\Delta t$ for $k = 0, 1, \ldots, N-1$, at which trades can be executed.

A *trading strategy*, Π, is defined as the sequence of trades $\{n_0, n_1, \ldots, n_{N-1}\}$, where n_k represents the number of shares executed at time-step t_k. For a strategy to be considered feasible, it must consist of non-negative trades and ensure the position is fully liquidated by the terminal time $t_N = T$. These constraints formally define the set of feasible strategies.

Definition 2.1 (Set of Feasible Trading Strategies [6])**.** The set of all feasible strategies, Θ, is defined as:

$$\Theta := \left\{ \Pi = \{n_0, n_1, \ldots, n_{N-1}\} : n_k \geq 0 \ \forall \ k \text{ and } \sum_{k=0}^{N-1} n_k = X \right\}$$

While Π defines the actions of a sell program, it is also useful to define the inventory state. Following a similar construction as in [1], we define the *trading trajectory*, $\{x_k\}_{k=0}^{N}$, where x_k for $k \in \{0, \ldots, N-1\}$ represents the inventory held at time t_k, immediately prior to the execution of the trade n_k, with $x_0 = X$ and $x_N = 0$, given a feasible strategy. In this sense the inventory state evolves according to the recursive relation $x_{k+1} = x_k - n_k$, from which it follows, the inventory at any time t_k, prior to the trade n_k, is given by:

$$x_k = X - \sum_{j=0}^{k-1} n_j = \sum_{j=k}^{N-1} n_j, \quad k = 0, 1, \ldots, N-1.$$

The cost of a trading strategy is then captured by the implementation shortfall, defined formally in 2.2.

Definition 2.2 (Implementation Shortfall)**.** For a strategy $\Pi \in \Theta$, the implementation shortfall is given by:

$$IS(\Pi) := X p_0^* - \sum_{k=0}^{N-1} n_k \bar{p}_k$$

where p_0^* is the asset mid-price at t_0 and $\bar{p}_k$ is the average price per share received for the trade n_k.

Execution prices are inherently stochastic, driven by natural price volatility, drift and the market impact generated by the traders actions. Consequently, IS is a random variable, and the objective of optimal execution is therefore to find the feasible strategy that best manages the trade-off between expected cost and cost uncertainty. This is formally stated as minimising a combination of the expected value and variance of the Implementation Shortfall:

$$\min_{\Pi \in \Theta} (\mathbb{E}[IS(\Pi)] + \lambda \operatorname{Var}([IS(\Pi)]) \tag{2.1}$$

where $\lambda \geq 0$ is a measure of the trader's risk-aversion ($\lambda = 0$ corresponds to risk-neutral, i.e. a TWAP schedule).

2.2.2 The Almgren & Chriss Approach

Almgren and Chriss work within a similar discrete-time framework as defined previously, for a pure sell program, though the trade n_k is taken to occur within the interval $[t_{k-1}, t_k]$ for $k = 1, 2, \ldots, N$, at an average execution rate $v_k = n_k/\Delta t$. It is then assumed that the asset price evolves according to the arithmetic random walk:

$$p_k^* = p_{k-1}^* + \sigma\sqrt{\Delta t}\xi_k - \Delta t g(v_k)$$

where σ is the volatility of the asset's price, $\xi_k \sim \mathcal{N}(0, 1)$ are i.i.d. random variables, and $g(v_k)$ is a function modelling the permanent impact of the traders actions.

To capture the cost of temporary impact, they define the average price per share received for the trade n_k to be:

$$\bar{p}_k = p_{k-1}^* - h(v_k)$$

where $h(v_k)$ is a function modelling the temporary price impact, which is assumed to remain only for the duration $[t_{k-1}, t_k]$. To formalise IS, the realised revenue for a given trajectory is computed as:

$$\sum_{k=1}^{N} n_k \bar{p}_k = X p_0^* + \sum_{k=1}^{N} (\sigma\sqrt{\Delta t}\xi_k - \Delta t g(v_k)) x_k - \sum_{k=1}^{N} n_k h(v_k)$$

The objective remains to then construct an efficient frontier of explicit solutions to the mean-variance minimisation problem (2.1), with the expectation and variance of IS defined by:

$$\mathbb{E}(IS) := \sum_{k=1}^{N} \Delta t x_k g(v_k) + \sum_{k=1}^{N} n_k h(v_k)$$

$$\text{Var}(IS) := \sigma^2 \sum_{k=1}^{N} \Delta t x_k^2$$

To obtain analytical solutions, Almgren and Chriss specify linear forms for the market impact functions[2]. Permanent impact is taken to be linear in the execution rate, $g(v_k) = \zeta v_k$ similarly, temporary impact is given by $h(v_k) = \epsilon + \eta v_k$. This results in an objective function (2.1) that is quadratic in the trading trajectory and hence is strictly convex for $\lambda \geq 0$, leading to a unique global solution. By solving the first-order conditions, they arrive at a linear second-order difference equation whose closed-form solution gives the optimal trading trajectory. In the continuous-time limit $\Delta t \to 0$, the optimal trading trajectory is given by:

$$x_k = \frac{\sinh(\kappa(T - t_k))}{\sinh(\kappa T)} X, \quad k = 0, \ldots, N \tag{2.2}$$

where

$$\kappa = \sqrt{\frac{\lambda \sigma^2}{\eta}}$$

is a key parameter determining the mean-variance trade-off–determined by the trader's specified level of risk-aversion λ.

The main limitation underpinning this construction is its dependance on strong, simplifying assumptions. Static or otherwise piecewise static strategies only hold to be globally optimal under the assumption that price returns are uncorrelated[3]. While the authors argue that the potential gains to be made by incorporating drift and serial price correlation are small, for a large initial position, they do so while maintaining the assumed linear market impact models. The tractability of analytical models can provide useful insights on average optimal paths, though an approach to optimal execution which is not agnostic to the complex LOB dynamics is inherently a dynamic state-dependent process.

[2]This is a modelling choice, used to simplify the explicit computation of optimal trajectories.
[3]It is a well documented fact that high-frequency markets posses serial price correlation [7].

2.3 Deep Reinforcement Learning

With the view that an optimal trading strategy is inherently a dynamic, state-dependant process, then at its core, the problem of optimal execution may be formulated as a *Markov Decision Process* (MDP)–the underlying framework of the Reinforcement Learning model. In this view, the optimal trading strategy becomes a problem of sequential decision-making under stochastic control.

The fundamental principle involves an RL *agent* taking trading actions within a simulated market *environment*. For each action, the agent receives a numerical reward conditioned upon how favourable the outcome is. The objective (2.1) can then be approximated by a discrete sum of scalar rewards, which the agent seeks to maximise over a given trajectory.

2.3.1 Elements of The Reinforcement Learning Model

The formal framework underpinning RL is the MDP, which provides a mathematical model for taking actions within a stochastic environment. Formally, a discrete-time MDP is defined by the tuple $(\mathcal{S}, \mathcal{A}, \mathbb{P}, \mathcal{R}, \gamma)$, where:

- $\mathcal{S}$ is the set of all possible states, $s \in \mathcal{S}$. A state defines a complete description of the current system and objective.
- $\mathcal{A}$ is the set of all possible actions, $a \in \mathcal{A}$.
- $\mathbb{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$ is the transition probability function, where $\mathbb{P}(s_{t+1}|s_t, a_t)$ provides the probability of transitioning to the state s_{t+1}, after having taken the action a_t in state s_t.
- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$ is the reward function, where $\mathcal{R}(s_t, a_t, s_{t+1})$ defines the scalar reward r_{t+1}, after having taken the action a_t, in state s_t and transitioned to the state s_{t+1}.

- $\gamma \in [0, 1)$ is the discount factor, used to determine the present value of all future rewards, where $\gamma < 1$ prioritises immediate rewards compared to future rewards.

The MDP formulation is predicated on the *Markov Property*, which asserts that $\mathbb{P}$ depends only on (s_t, a_t), with no hidden dependence on the previous history. While the underlying dynamics of LOBs are known to be non-Markovian, the objective in practice is then to construct a state representation s_t that is sufficiently informative to render this assumption a valid approximation[4].

In this framework an agent can interact with the environment through its *policy*, $\pi(a_t|s_t)$, which maps observed states to a probability distribution over the action space. Through interacting with the environment, the policy generates an *episode*, τ, which is defined by the sequence of realised states, actions, and rewards received over the horizon:

$$\tau = \langle s_0, a_0, r_1, s_1, a_1, r_2, \ldots, s_{T-1}, a_{T-1}, r_T, s_T \rangle$$

The agent's goal is to then learn a policy that maximises the expected *return*. The return, G_t, is the discounted sum of rewards from time-step t onwards:

$$G_t = \sum_{j=1}^{T-t} \gamma^{j-1} r_{t+j} \tag{2.3}$$

From this follows the formal definition of the agent's objective and the concept of an optimal policy.

Definition 2.3. The *objective function*, $J(\pi)$, is the expected return from the initial state, given the agent follows the policy π:

$$J(\pi) = \mathbb{E}_{\tau \sim \pi}[G_0]$$

[4]Within a data-driven simulator, satisfying the Markov property requires the agent's state representation, s_t, to contain all information used by the simulator's transition mechanism (e.g., the features used in the k-NN distance metric) to determine the next state.

An *optimal policy*, π^*, is any policy that maximises this objective function:

$$\pi^* = \arg\max_{\pi} J(\pi)$$

To achieve this objective, most RL algorithms rely on estimating the expected return, or "value", of being in certain states or taking certain actions. This method of evaluation is captured by a set of value functions used for policy refinement.

Definition 2.4 (State-Value function)**.** The state-value function, $V^\pi(s)$, defines the expected return when starting from a state, $s \in \mathcal{S}$, and following the policy π thereafter. It quantifies how favourable a given state is under π:

$$V^\pi(s) := \mathbb{E}_\pi[G_t | s_t = s]$$

Definition 2.5 (Action-Value function)**.** The action-value function, $Q^\pi(s, a)$, defines the expected return after taking an action $a \in \mathcal{A}$ in a state $s \in \mathcal{S}$ and subsequently following the policy π. It quantifies how favourable a given state-action pair is under π:

$$Q^\pi(s, a) := \mathbb{E}_\pi[G_t | s_t = s, a_t = a]$$

Definition 2.6 (Advantage function)**.** The advantage function, $A^\pi(s, a)$, measures the relative benefit of taking a specific action a in state s, compared to the average action prescribed by π in that state:

$$A^\pi(s, a) := Q^\pi(s, a) - V^\pi(s)$$

2.3.2 DRL for Optimal Execution

Having defined the MDP framework and the RL objective (2.3), we now consider its application to the problem of optimal execution. The methods discussed here fall within the paradigm of *Deep Reinforcement Learning* (DRL), in which the policy and/or value functions are parameterised using *Deep Neural Networks* (DNNs).

There are several distinctions that define a given implementation, these include; the method of environment (LOB) simulation, the choice of RL algorithm for policy optimisation, and the specific MDP formulation–that is, the choice of state representation, action space and reward function.

Regarding the choice of RL algorithm, a variety of options exist which can be broadly categorised into three classes. *Policy-based* methods directly optimises the policy π_θ (the Actor-Network) through gradient ascent on the objective function (2.3). While effective for learning stochastic policies, they can suffer from high variance in their gradient estimates [8, Ch. 13]. In contrast, *value-based* methods optimise the value functions Q_φ and V_φ (the Critic-Network), through gradient descent, from which the optimal policy is implicitly derived. These methods often provide lower variance, though converge to deterministic policies, suboptimal in environments requiring stochasticity. Finally, hybrid *Actor-Critic* methods leverage the low-variance estimates from a learned critic-network to inform updates in the actor-network, enabling the stable convergence of a stochastic policy.

In a recent implementation, Hafsi & Vittori [9], approach the problem of optimal execution with a value-based DRL agent, specifically using the DQN algorithm [10]. Their MDP formulation consists of a state space representation with two *private* elements[5]–the percentage of remaining inventory x_k/X and time $(T-t_k)/T$–and three LOB state metrics:

[5]These so-called private elements are a consistent requirement in many DRL implementations, providing the agent with information on the current state of the liquidation task.

volume imbalance across five active price levels and the best bid/ask prices. The agent's action space is taken as a discrete set of five execution sizes.

Critically, their per-step reward function is designed to minimise the expected implementation shortfall:

$$r_{t+1} = \underbrace{n_k \times (\bar{p}_k - p_0^*)}_{IS} \quad - \quad \underbrace{\alpha d_t}_{\text{depletion penalty}} \quad , \quad \alpha > 0$$

with the added depletion penalty term where, d_t, is the number of active price levels consumed by the order, which intelligently discourages large orders during periods of low liquidity. However, the reduction of IS variance is somewhat weakly implicit in this design. With an inherent incentive to clear the position faster as to receive zero, non-negative rewards, though no explicit minimisation of $\lambda \operatorname{Var}(IS)$ is considered.

In contrast to this value-based approach, several actor-critic implementations have also been considered. For instance, Lin & Beling [11] employ a PPO algorithm, while Micheli & Monod [12] implement a DDPG algorithm. Although these works showcase equally viable RL algorithms and principled MDP formulations, a common thread in these works is their reliance on simulation environments with strong underlying assumptions about market dynamics. Lin & Beling, for example assume a fully resilient market, in which, only temporary price impact is considered with no remaining price depression in the next time step. Whereas Micheli & Monod model the price process as a Brownian motion with a transient impact kernel. This reliance on simplifying parametric models provides clear motivation for developing a DRL implementation to optimal execution, that learns within a non-parametric environment, ensuring the learnt policy is free from restrictive modelling biases–essential for a robust policy.

2.4 Limit Order Book Simulation

The successful application of DRL to optimal execution necessitates a reliable method of LOB simulation. In particular, the simulator must be able to replicate many of the key

statistical properties and stylised facts observable in historical data. Furthermore, the simulator must enable accurate counterfactual dynamics, that is, the ability to accurately model the market's endogenous response to the agent's trades. Relying solely on historical data risks overfitting to a single market realisation and is fundamentally a static replay, incapable of modelling the market's reactive dynamics.

Several paradigms exist for LOB simulation, each presenting a different trade-off between mathematical tractability, computational feasibility and fidelity. For a comprehensive review of simulation methods we refer the reader to [3]. While achieving varying levels of fidelity, many of the established approaches suffer from specific drawbacks; these include a reliance on strong modelling assumptions that risk misrepresenting market dynamics, the need for complex calibration procedures and being computationally intensive–a critical limitation for DRL implementations.

The K-Nearest Neighbour (K-NN) resampling method, recently proposed by Giegrich et al. [4], offers a compelling solution to these challenges. Their approach introduces a statistical learning method for LOB simulation, which is inherently non-parametric, is computationally efficient and does not require complex calibration or optimisation. The core idea is predicated on similar LOB states having similar subsequent evolutions. By resampling historical transitions, from similar historical states, the simulator can generate many plausible market realisations while remaining empirically grounded and implicitly modelling the complex and endogenous dynamics of the LOB.

2.4.1 K-NN Resampling

Here we outline the general simulation method developed in [4]. For computational tractability, the simulator operates within a discrete-time framework similar to the construction outlined in Subsection 2.2.1.

The method assumes the LOB's state can be sufficiently captured by a given instan-

taneous state representation, Z_{t_k}, which then evolves as a discrete random dynamical system. Central to this approach is a "transition library", $\mathcal{D}$, which contains the set of historical LOB state-price pairs and their subsequent transitions:

$$\mathcal{D} = \{(Z_{t_k}, p^*_{t_k}), (Z_{t_k+\Delta t}, p^*_{t_k+\Delta t})\}_{k=0}^{M}$$

Where Z_{t_k} is the historical LOB state at a given time t_k and $Z_{t_{k+\Delta t}}$ is the state that followed. Simulations are then initialised in a given historical state $(Z_{t_0}, p^*_{t_0})$, against which an agent can act on, moving it to a post-trade state $(\tilde{Z}_{t_0}, \tilde{p}^*_{t_0})$. Updates are then determined in the following manner:

1. **Similarity Search**: The feature vector $\tilde{Z}_{t_0}$ is compared against all historical states in $\mathcal{D}$ and the K "nearest neighbours"–the historical states most similar to $\tilde{Z}_{t_0}$ according to a pre-defined distance metric–are identified.

2. **Stochastic Resampling**: One of these K neighbours is selected uniformly at random.

3. **State Evolution**: The historical transition experienced in the historically matched state is then used to evolve the simulated state.

This resampling mechanism is inherently Markovian in its dependence on i.i.d. transitions of instantaneous state representations. However, as noted in [4], the non-Markovian nature of true LOB dynamics can be approximated by including features of past evolution (e.g., recent price returns, see 3.1.2) within the state vector Z_{t_k}. For a full treatment of the K-NN simulation method see section 3.2.

Chapter 3

Methodology

This chapter details the construction of a LOB trading simulator using the K-NN resampling method discussed in subsection 2.4.1, with the subsequent development and implementation of a DRL agent for the task of optimal liquidation. We begin by outlining the data set used and the pre-processing steps applied to build the simulator's transition library $\mathcal{D}$, which forms the core of the environment for training our DRL agent.

3.1 The Data Set

To develop our trading environment, we selected a high-liquidity stock to aid the data-driven nature of the K-NN simulator. Specifically, our study focuses on Microsoft Corporation (MSFT) stock. The transition library, $\mathcal{D}$, was constructed from a data set of 26 trading days of MSFT LOB data sourced from Databento [13], covering the period from 2025-02-10 to 2025-03-18.

The specific schema used is MBP-10 (Market-By-Price with 10 levels), which provides the event-driven historical data stream, capturing the state of the LOB across 10 active price levels on both the bid and ask sides of the book. Updates in the book are triggered by market events (the arrival of new or modifications to existing orders) captured at nanosecond-level precision. A given event recorded in the book defines the details of the event and the resulting state of the book. To avoid periods of low liquidity and unchar-

acteristic market dynamics, we only consider events within standard NASDAQ trading hours (09:30 to 16:00 EST).[1]

3.1.1 Pre-Processing

To align the event-driven data with our discrete-time framework, the data was downsampled into 1-second snapshots. This frequency was chosen as a balance between capturing high-resolution market dynamics while maintaining computational tractability. For each 1-second interval, the last observed LOB state for a given bin was taken as the representative snapshot for that time-step. In cases where an interval contained no market events, the LOB state from the previous second was forward-filled, under the assumption that the book remained static. This provided us with a data set of 26 trading days (6.5 hours each) with 23400 observations per day, and 608400 in total.

For computational feasibility we adopt an aggregated view of the LOB, as modelling the exact matching sequence outlined in section 2.1 would have resulted in a moderately high dimensional problem, which would require tracking the queued positions of all limit orders. Second, and more fundamentally, such a detailed model is unnecessary for our problem domain, as the agent is constrained to a pure sell program using only market orders. Hence for this study we assume the LOB to be a discrete set of states separated by unobserved evolution. For this reason we can discard many of the event details and focus on the resulting state, such as the prices of each level $p^b_{t,i}$, $p^a_{t,i}$ and the respective volumes at each level $V^b_{t,i}$, $V^a_{t,i}$.

For later evaluation we partition the data set chronologically into an approximate 80-20 split, with the first 20 days used to build $\mathcal{D}$ and the subsequent 6 days held out for testing.

[1] The original dataset is given in UTC, and includes a daylight saving shift on 03-10, this was explicitly accounted for in our data preparation.

3.1.2 Transition Library Construction

In the implementation by Giegrich et al. [4], a fixed tick-grid centred about the dividing (or mid) price is employed, with the state representation, Z_t, capturing the order volumes populating the discrete levels. This design ensures a constant state size and achieves a price-invariant representation, in which it is assumed that the LOB's evolution depends on its relative shape, not its absolute price level.

However, in our MSFT dataset, the spacing between active price levels was found to be highly variable, with some periods having the 10 price levels tightly clustered, with single-tick ($0.01) spacing, while in others they are significantly more dispersed. A fixed tick-based grid would therefore be impractically large and sparse to capture all active levels in the dataset. Thus to retain all of our data and to capture the relative book shape while remaining price-invariant, we include the price gaps between adjacent active levels.

Building on this, we now formally define the construction of our state vector Z_t, with elements that can be categorised into three groups:

1. **LOB Shape Features**: These features provide a price-invariant representation of the current LOB structure, consisting of:

   - **Normalised Volumes**: The volume at each of the $L = 10$ bid and ask levels, normalised by the total volume on its respective side of the book. This captures the relative distribution of liquidity. For bid levels $i \in \{1, 2, \ldots, L\}$:

   $$Q^b_{t,i} = \frac{V^b_{t,i}}{\sum_{j=1}^{L} V^b_{t,j}} \quad \text{(and analogously for } Q^a_{t,i}\text{)}$$

   This results in a vector of $2L$ normalised volumes.

   - **Price Gaps**: The price differences between adjacent active levels on both the bid and ask sides. This captures the relative price structure, which would

otherwise be infeasible on a fixed-tick grid. For levels $i \in \{1, 2, \ldots, L-1\}$:

$$\delta^b_{t,i} = p^b_{t,i} - p^b_{t,i+1}$$

$$\delta^a_{t,i} = p^a_{t,i+1} - p^a_{t,i}$$

This produces a vector of $2(L-1)$ price gaps.

- **Bid-Ask Spread**: As defined in Table 2.1, this provides a key measure of liquidity:

$$w_t = p^a_{t,1} - p^b_{t,1}$$

2. **Momentum Features**: To account for the non-Markovian nature of the LOB, we incorporate features that describe the recent price trajectory, specifically we include the log returns of the mid-price over several look-back windows, for $h \in \{5, 30, 120\}$:

$$\rho^{(h)}_t = \ln(p^*_t) - \ln(p^*_{t-h})$$

3. **Temporal Features**: We index states according to their time of day as to capture the intra-day seasonalities in liquidity and volatility. Specifically, the trading day is partitioned into three discrete regimes: market open, mid-day, and market close. This categorical feature is one-hot encoded into a binary vector, allowing the model to condition its searches on the time of day:

$$C_t = \begin{cases} 0, & t \in [\text{09:30:00}, \text{10:30:00}) (\text{market open}) \\ 1, & t \in [\text{10:30:00}, \text{15:00:00}) (\text{mid-day}) \\ 2, & t \in [\text{15:00:00}, \text{16:00:00}) (\text{market close}) \end{cases}$$

and its one-hot encoded vector, $R_t \in \{0, 1\}^3$, is defined by its components:

$$(R_t)_i = \mathbf{1}_{\{C_t = i\}} \quad \text{for} \quad i \in \{0, 1, 2\}$$

To ensure the rolling windows calculated in momentum features are correctly confined to within day observations, we group by day when deriving state features. Furthermore, the required look-back window renders the first 119 seconds of each day unusable. Similarly, the final observation of each day is discarded as it has no subsequent $t+1$ state to form a transition. This leaves the final 20 day set to have 465600 observations.

The features described above form the basis of our transition library, $\mathcal{D}$. As introduced in subsection 2.4.1, this library is conceptually a collection of historical state transitions. We now provide a more formal construction based on our specific feature engineering.

The library is a set of tuples, $\mathcal{D} = \{(Z_t, O_{t+1})\}_{t=0}^{M}$, where Z_t is a concatenation of the LOB shape, momentum and temporal features representing the LOB at time t, and O_{t+1} is the outcome vector, containing the full shifted set of raw LOB values: $\{p^a_{t+1,i}, p^b_{t+1,i}\}_{i=1}^{L}$, $\{V^a_{t+1,i}, V^b_{t+1,i}\}_{i=1}^{L}$ and the mid-price p^*_{t+1}.

For the K-NN similarity search, the raw feature vector Z_t is not used directly. Instead, it undergoes a transformation pipeline, Φ, consisting of standardisation and Principal Component Analysis (PCA)–designed to prepare the data for a meaningful distance metric. First, each feature $z_{t,j} \in Z_t$ is standardised to have zero mean and unit variance:

$$z^{'}_{t,j} = \frac{z_{t,j} - \mu_j}{\sigma_j}$$

where μ_j and σ_j are the mean and standard deviation of the j-th feature, calculated over the 20 day set. Let this standardisation operation be denoted by $\mathcal{N}$. The resulting standardised vector is then $Z^{'}_t = \mathcal{N}(Z_t)$, with 45 dimensions, which is now more suitable for distance-based similarity searches, as elements with larger scales no longer dominate this search.

The standardised vector $Z^{'}_t$ is then projected onto its first $d = 35$ principal components. Let this PCA projection be denoted by the operator $\mathcal{P}$. This step reduces noise and

computational complexity while retaining approximately 90% of the explained variance. The final transformed state vector, Y_t, used for the similarity search is therefore given by:

$$Y_t = \Phi(Z_t) = (\mathcal{P} \circ \mathcal{N})(Z_t).$$

3.2 Simulation Environment

Having defined the feature vector, Z_t, and its final transformed state, Y_t, we now specify the mechanics of the LOB simulation environment. This section details the problem-specific environment where an agent is constrained to placing sell market orders and in which $\Delta t = 1$. We first describe how simulation episodes are initialised. We then detail the two primary stages of a state transition: the execution of an agents trade, followed by the stochastic evolution of the market.

Here we denote simulated states by $\hat{O}$, intermediate post-trade states with $\tilde{O}$ and historically matched states resampled from K-NN with $\bar{O}$.

Firstly, a simulation episode is initialised by selecting a raw historical state uniformly at random from the 20 day set, providing the initial simulated state:

$$\hat{O}_{t_0} = (\{\hat{p}^a_{t_0,i}, \hat{p}^b_{t_0,i}, \hat{V}^a_{t_0,i}, \hat{V}^b_{t_0,i}\}_{i=1}^{L},\ \hat{p}^*_{t_0})$$

A given representation of this state, s_{t_0}, is then passed to the agents policy which outputs an action corresponding to the shares, $n_0 \geq 0$, that are to be executed against $\hat{O}_{t_0}$.

To prevent poor out-of sample states for K-NN similarity searches, we limit the volume that any given market order, n_k, can consume. Specifically, we define the maximum

consumable liquidity, $\mathcal{L}_{t_k}$, to be a fraction of the total bid volume:

$$\mathcal{L}_{t_k} = \kappa \sum_{i=1}^{L} \hat{V}^b_{t_k,i}$$

where $\kappa \in [0, 1)$, is a tuneable parameter that can define boundaries within which the agent can operate. The number of shares that are actually executed is therefore given by:

$$n^{'}_k = \min(n_k, \lfloor \mathcal{L}_{t_k} \rfloor)$$

We can then formally define a market order function, g_M, that acts on the bid side volume, sequentially moving each level's volume to zero until the order is fulfilled or the maximum consumable liquidity cap is met:

$$g_M\left(\{\hat{V}^b_{t_k,i}\}^L_{i=1}, n^{'}_k\right) = \{\tilde{V}^b_{t_k,i}\}^L_{i=1} = \left\{\hat{V}^b_{t_k,i} - \min\left(\max\left(n^{'}_k - \sum_{j=1}^{i-1} \hat{V}^b_{t_k,j},\, 0\right),\, \hat{V}^b_{t_k,i}\right)\right\}^L_{i=1}$$

It also follows that the post trade mid-price, $\tilde{p}^*_{t_k} \neq \hat{p}^*_{t_k}$, if and only if $n^{'}_k \geq \hat{V}^b_{t_k,1}$, in which case, the post trade mid-price is found from using the highest bid price with non-zero volume, the same applies for $\tilde{w}_{t_k}$.

The resulting intermediate post-trade state, $\tilde{O}_{t_k}$, is a composite of unchanged and potentially changed components from the pre-trade state $\hat{O}_{t_k}$, captured by:

$$\tilde{O}_{t_k} = (\{\hat{p}^a_{t_k,i}, \hat{p}^b_{t_k,i}, \hat{V}^a_{t_k,i}, \tilde{V}^b_{t_k,i}\}^L_{i=1},\ \tilde{p}^*_{t_k})$$

Before the state transition can occur we must then apply the same feature derivation and transformation pipeline to arrive at $\hat{Y}_{t_k}$. First the feature vector is constructed:

$$\hat{Z}_{t_k} = g_Z(\tilde{O}_{t_k})$$

Followed by the transition pipeline:

$$\hat{Y}_{t_k} = \Phi(\hat{Z}_{t_k}) = (\mathcal{P} \circ \mathcal{N})(\hat{Z}_{t_k}).$$

Following this transitions can then take place, in [4] the authors find the K nearest neighbours using a Euclidean distance, though it is noted several other metrics can be employed, such as a weighted Euclidean norm or an Wasserstein metric. They then proceed to select one of these K neighbours uniformly at random. We propose an alternate approach that weights neighbours according to their Euclidean distance similarity. Specifically, we employ an inverse rank weighting, in which the K neighbours are ordered by increasing Euclidean distance from $\hat{Y}_{t_k}$, yielding indices, $\{j_\ell\}_{\ell=1}^{K}$, where $\ell = 1$ is the closest (rank 1), $\ell = 2$ is rank 2, etc. The weight for the ℓ-th neighbour is then given by:

$$W_\ell = \frac{1}{\ell}$$

Neighbours are then sampled according to the discrete probability distribution:

$$\{P_\ell\}_{\ell=1}^{K}, \quad \text{where} \quad P_\ell = \frac{W_\ell}{\sum_{m=1}^{K} W_m}$$

This promotes more transitions from the most historically similar states, while also allowing for lower similarity transitions.

From the sampled rank ℓ, the matched historical outcome vector, $\bar{O}_{t+1} = g_P(\hat{Y}_{t_k}, \mathcal{D})$, is retrieved[2], where $g_P(\hat{Y}_{t_k}, \mathcal{D})$ performs the K-NN search, computes the ranks and probabilities as above, samples ℓ and returns the historical outcome. To finalise the simulated state update, the volumes are taken as:

$$\{\hat{V}^a_{t_k+1,i}, \hat{V}^b_{t_k+1,i}\}_{i=1}^{L} \leftarrow \{\bar{V}^a_{t+1,i}, \bar{V}^b_{t+1,i}\}_{i=1}^{L}$$

[2]Where t denotes the historical time index of the matched neighbour, distinct from the simulation time t_k.

with the mid-price update incorporating the historically observed price evolution from the impacted post-trade state:

$$\hat{p}^*_{t_k+1} \leftarrow \tilde{p}^*_{t_k} + (\bar{p}^*_{t+1} - \bar{p}^*_t)$$

and finally, the updated prices are the historical outcome prices re-centred about our simulated mid-price:

$$\{\hat{p}^a_{t_k+1,i}, \hat{p}^b_{t_k+1,i}\}_{i=1}^L \leftarrow \hat{p}^*_{t_{k+1}} + (\{\bar{p}^a_{t+1,i}, \bar{p}^b_{t+1,i}\}_{i=1}^L - \bar{p}^*_{t+1}).$$

A full treatment of the simulation algorithm can be found in Algorithm 1, with further implementation details covered in subsection 3.2.1.

Algorithm 1 LOB Simulation Enviroment

initialise in historical state: $\hat{O}_{t_0} = (\{\hat{p}^a_{t_0,i}, \hat{p}^b_{t_0,i}, \hat{V}^a_{t_0,i}, \hat{V}^b_{t_0,i}\}_{i=1}^L,\ \hat{p}^*_{t_0})$
for t_k from $k = 0$ to T:
    agent acts: $\pi(a_{t_k}|s_{t_k}) \rightarrow n_{t_k}$
    shares available to trade: $n'_k = \min(n_k, \lfloor \mathcal{L}_{t_k} \rfloor)$
    execute market sell: $\tilde{O}_{t_k} = g_M\left(\{\hat{V}^b_{t_k,i}\}_{i=1}^L, n'_k\right)$
    state feature derivation: $\hat{Z}_{t_k} = g_Z(\tilde{O}_{t_k})$
    transformation pipeline: $\hat{Y}_{t_k} = \Phi(\hat{Z}_{t_k})$
    re-sample historical outcome: $\bar{O}_{t+1} = g_P(\hat{Y}_{t_k}, \mathcal{D})$
    state update $\hat{O}_{t_{k+1}}$:
        $\{\hat{V}^a_{t_k+1,i}, \hat{V}^b_{t_k+1,i}\}_{i=1}^L \leftarrow \{\bar{V}^a_{t+1,i}, \bar{V}^b_{t+1,i}\}_{i=1}^L$
        $\hat{p}^*_{t_k+1} \leftarrow \tilde{p}^*_{t_k} + (\bar{p}^*_{t+1} - \bar{p}^*_t)$
        $\{\hat{p}^a_{t_k+1,i}, \hat{p}^b_{t_k+1,i}\}_{i=1}^L \leftarrow \hat{p}^*_{t_{k+1}} + (\{\bar{p}^a_{t+1,i}, \bar{p}^b_{t+1,i}\}_{i=1}^L - \bar{p}^*_{t+1})$
        $\hat{R}_{t_{k+1}} \leftarrow \bar{R}_{t+1}$
    update log returns: $\hat{\rho}^{(h)}_{t_{k+1}}$
end for

This construction allows for LOB simulation with or without agent trades, in which market impact and state evolution is modelled implicitly through empirically realised transitions. This approach also allows for K^T possible market realisations given any initial state.

3.2.1 LOB Simulation Specifics

Here we outline some important details about our implementation, regarding parameter selection and specific simulation assumptions made.

Initialisation of Momentum Features: To ensure the momentum features, $\hat{\rho}_t^{(h)}$, are accurately computed from the start of each episode, we load the preceding 119 seconds of historical mid-price data to populate the lookback windows. This ensures we initialise to a state with a valid price history. However, in the case when an episode is initialised within the first 120 seconds of a trading day, the available history extends only to the first observation at market open, beyond which we backwards-fill the opening mid-price, assuming a static price prior to market open.

Time Regime Updates: The temporal feature vector, $\hat{R}_{t_k}$, is not evolved chronologically within a simulation episode. Instead, at each step, the simulator assumes the regime of the matched historical state: $\hat{R}_{t_{k+1}} \leftarrow \bar{R}_{t+1}$. We reason this choice by positing the inclusion of R_t in the feature space serves to identify regions of temporally similar states, which often exhibit comparable intra-day characteristics. This approach prevents the algorithm from limiting g_P from missing a group of highly similar states in a different time regime.

Parameter Selection: The parameters chosen here were found via grid search over the 20 day build set, selecting the parameters which optimised a set of pre-defined fidelity metrics outlined in section 4.1. The selected values are:

- **Number of Neighbours,** K: Set to 30. The selection of K is determined by a bias-variance trade-off, a common heuristic value which is employed in [4] is 20, though the use of our inverse rank weighting limits variance, necessitating a larger set of neighbours.
- **Liquidity Cap** κ: Set to 0.9. This was found to be the highest value providing

the agent with the most freedom, before creating un-realistic post-trade states that fall outside the distribution of our historical data, which consequently result in poor transitions.

3.3 DRL Implementation

In this section we outline our specific MDP formulation of the optimal execution problem, and the approach taken to policy optimisation. The agent described here will be tasked with selling 20000 MSFT shares over a 30 minuet horizon ([0,1800]) within the environment described in Algorithm 1.

3.3.1 MDP Formulation

As detailed in subsection 2.3.1, there are three key elements in formulating a problem as a MDP. These are, the observation space $s \in \mathcal{S}$, the action space $a \in \mathcal{A}$ and the reward function $\mathcal{R}$. Here we propose a formulation which seeks to approximate the mean-variance minimisation objective 2.1 of optimal execution, within our previously constructed trading environment.

1. **Action Space, $\mathcal{A}$**: In optimal liquidation, through a pure sell program, actions taken by the agent must translate into an integer share value, n_k, that is attempted for execution against $\hat{O}_{t_k}$. For sample efficiency during optimisation, we restrict ourselves to a discrete action space. Specifically, we employ a discrete set of 13 actions a:

$$\mathcal{A} = \{0\%, 0.5\%, 1\%, 2\%, 3\%, 5\%, 7\%, 10\%, 15\%, 25\%, 35\%, 50\%, 100\%\}$$

which correspond to the percentage of remaining inventory, x_k, that is attempted for execution at t_k, where the conversion to shares is given by:

$$n_k = \lfloor a_{t_k} \cdot x_k \rfloor$$

This design allows for finer control in placing small child orders, including the ability to wait (0%) for more favourable conditions, while becoming more coarse over the aggressive actions. This set-up also ensures the agents policy is scale-invariant to the initial position size X.

2. **Observation Space, $\mathcal{S}$**: The observation space constitutes the set of environment variables in which the agent conditions its actions and rewards upon. We aim for a parsimonious representation, while seeking to approximate the Markov property. Specifically we include two private task-oriented variables and seven LOB state representative variables, chosen for their relevance to the optimal execution problem:

   - **Remaining Inventory**: The percentage of inventory remaining at time t_k, informing the agent on its current progress, given by:

   $$\frac{x_k}{X}$$

   - **Remaining Time**: The percentage of remaining time at t_k, given by:

   $$\frac{T - t_k}{T}$$

   - **Momentum Features**: To aid the approximation of the Markov property, we include the non-Markovian mid-price returns from which K-NN searches are conditioned upon. In particular, we include the log returns over two lookback windows, $h \in \{5, 120\}$:

   $$\hat{\rho}_{t_k}^{(h)} = \ln(\hat{p}_{t_k}^*) - \ln(\hat{p}_{t_k-h}^*)$$

   - **Bid-Ask Spread**: A key determinant of available liquidity, also used in the K-NN resampling:

   $$\hat{w}_{t_k} = \hat{p}_{t_k,1}^a - \hat{p}_{t_k,1}^b$$

   - **Order Book Imbalance**: We take the volume imbalance measure introduced

by Cartea et al. [14] and extend it over the top five levels, this measure was shown to predict the sign and magnitude of price changes:

$$\text{OBI}^{(5)} = \frac{\sum_{i=1}^{5} \hat{V}^b_{t_k,i} - \sum_{i=1}^{5} \hat{V}^a_{t_k,i}}{\sum_{i=1}^{5} \hat{V}^b_{t_k,i} + \sum_{i=1}^{5} \hat{V}^a_{t_k,i}} \in [-1, 1]$$

- **Relative Available Bid Depth**: This variable provides the agent with a clear measure of the current tradable volume, normalised by the initial position:

$$\frac{\mathcal{L}_{t_k}}{X}$$

- **Available Spread**: This captures the price difference across the top bid price down to the price level determined by $\mathcal{L}_{t_k}$. Providing a measure of how favourable the current price density is, formally:

$$\text{for} \quad m = \min\left\{ j \in \{1, \dots, L\} : \sum_{i=1}^{j} \hat{V}^b_{t_k,i} \geq \mathcal{L}_{t_k} \right\} \quad \text{then,}$$

$$w'_{t_k} = \hat{p}^b_{t_k,1} - \hat{p}^b_{t_k,m}$$

- **Normalised Bid-Side Slippage**: This feature calculates the normalised price difference across the best bid and the *Volume Weighted Average Price* (VWAP), providing a measure for the immediate shortfall across the top five levels, normalised by the mid price:

$$\frac{\hat{p}^b_{t_k,1} - \frac{\sum_{i=1}^{5} \hat{p}^b_{t_k,i} \cdot \hat{V}^b_{t_k,i}}{\sum_{i=1}^{5} \hat{V}^b_{t_k,i}}}{\hat{p}^*_{t_k}}$$

3. **Reward Function, $\mathcal{R}$**: Reward functions, as in [9], often seek to minimise the expected IS, though often lack explicit consideration of minimising IS variance. To account for this, we employ a dense per-step reward function which seeks a policy that minimises temporary impact while explicitly addressing variance risk. First we define the revenue received, $\mathcal{C}_{t_k}$, for trade n'_k, to be the sum of the product

of shares executed at the i-th level, n_i, and the respective price $\hat{p}^b_{t_k,i}$:

$$\mathcal{C}_{t_k} = \sum_{i=1}^{L} n_i \cdot \hat{p}^b_{t_k,i}$$

The immediate shortfall at t_k for trade $n_k^{'}$ is then:

$$n_k^{'}\hat{p}^*_{t_k} - \mathcal{C}_{t_k}$$

This constitutes our temporary impact term[3]. To then incorporate a proxy for variance minimisation, we first approximate the price variance, $\sigma^2_{t_k}$, using a weighted average of realised variance estimates (under the simplifying assumption of zero drift over short windows). Specifically, for the look-back windows, $\{h_m\}_{m=1}^3 = \{5, 30, 120\}$, and the corresponding weights $\{w_m\}_{m=1}^3 = \{0.8, 0.15, 0.05\}$ (prioritising recency), we take the approximate price variance as:

$$\sigma^2_{t_k} = \sum_{m=1}^{3} w_m \cdot \frac{(\hat{\rho}^{(h_m)}_{t_k})^2}{h_m}$$

Then mirroring the variance term from the Almgren & Chriss model, we define the risk term:

$$\mathcal{V} = \lambda_{\text{risk}} \cdot (\sigma^2_{t_k}) \cdot (\hat{p}^*_{t_k})^2 \cdot (x_{k+1})^2 \cdot (T - t_k)$$

where λ_{risk} is a tuneable risk-aversion parameter, with which we define $\lambda_{\text{risk}} = 5 \times 10^{-8}$. The per step reward is then:

$$r_{t_{k+1}} = -(n_k^{'}\hat{p}^*_{t_k} - \mathcal{C}_{t_k}) - \mathcal{V}$$

To ensure the liquidation task is completed, we include a large terminal penalty for

[3]Due to the lack of available methods for accurately modelling the future path-dependent permanent impact, and to maintain a dense, per-step reward signal, we focus on minimising the trade-off between temporary impact and timing risk, making the simplifying assumption that permanent future impact is approximately dependant on the initial portfolio size, not the strategy Π–see section 4.2

an episode with $x_T \neq 0$, specifically we set:

$$r_T = -10(\hat{p}^*_{t_0} \cdot x_T)$$

as a form of reward shaping.

3.3.2 Policy Optimisation

With the MDP elements formalised, we can now proceed with the DRL policy optimisation algorithm. In particular, we consider the implementation of Proximal Policy Optimisation (PPO) [5], using separate Actor-Critic networks. As discussed in subsection 2.3.1, the agents policy is a mapping from observed states $s \in \mathcal{S}$ to a probability distribution over actions $a \in \mathcal{A}$. The agent operates within Algorithm 1, generating an episode, τ, of realised states, actions and rewards. PPO iteratively collects episodes under a policy and separately updates the parameterised Actor network $\pi_\theta(a|s)$ and the Critic network $V_\varphi(s)$, which provides state-value estimates informing policy updates.

PPO utilises advantage estimates (Definition 2.6), $\hat{A}$, using the critic's state-value estimates to inform policy updates. Specifically, to avoid instability from direct gradient ascent on the objective function (Definition 2.3), $\nabla_\theta J(\theta)$, PPO provides a clipped "surrogate" objective function, which approximates the policy gradient:

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

where, $r_t(\theta)$ (distinct from the scalar reward r_t), is the probability ratio:

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

defining how much more or less likely a given action, a_t, is to be taken under the new policy versus the old policy under which the episode was collected, and ϵ is the clipping parameter–limiting the amount by which sequential policies can differ after up-

dates. For example, given a favourable action a_t, $\hat{A}_t > 0$, then the objective becomes: $\min(r_t(\theta)\hat{A}_t, (1+\epsilon)\hat{A}_t)$. Furthermore, to encourage policy space exploration (limiting early convergence to a local optimum), an entropy regularisation term is included:

$$H(\pi_\theta(a|s_t)) = -\sum_{a\in\mathcal{A}} \pi_\theta(a|s_t)\ln(\pi_\theta(a|s_t))$$

giving the full actor loss:

$$L(\theta) = -L^{\text{CLIP}}(\theta) - c_2 H(\pi_\theta)$$

where, $c_2 > 0$, is an entropy scaling parameter, which we anneal linearly over policy optimisation, allowing for early exploration while later stabilising for convergence.

The critic network learns the state-value function by minimising the Mean Squared Error (MSE) between its value estimates $V_\varphi(s_t)$ and the empirical returns (Eq. 2.3) calculated from the episode, $\hat{G}_t$, via gradient descent on the critic loss:

$$L(\varphi) = \hat{\mathbb{E}}[(V_\varphi(s_t) - \hat{G}_t)^2].$$

For advantage estimation we employ Generalised Advantage Estimation (GAE) [15]–a commonly used method for its balancing of variance and bias in its advantage estimates. The approach takes an exponentially weighted average of all k-step estimates:

$$\hat{A}_t = \sum_{k=0}^{T-t-1} (\gamma\lambda)^k \cdot (r_{t+1+k} + \gamma V_\varphi(s_{t+1+k}) - V_\varphi(s_{t+k}))$$

or equivalently the backwards recursive form as used in practice from $t = T-1 \to 0$:

$$\hat{A}_t = \delta_t + \gamma\lambda\hat{A}_{t+1}$$

where $\lambda \in [0, 1]$, is the parameter controlling the bias-variance trade-off, $\hat{A}_T = 0$ and δ_t,

is the Temporal Difference (TD) error at step t:

$$\delta_t = r_{t+1} + \gamma V_\varphi(s_{t+1}) - V_\varphi(s_t)$$

The optimisation process is then structured as an iterative loop consisting of data collection and policy updates. Specifically, iterations start by collecting a large batch of complete episodes, τ, under the current policy. Following this the empirical returns, $\hat{G}_t$, and Advantage estimates, $\hat{A}_t$ are calculated for every time-step and are standardised across the batch to have zero mean and unit variance for stability. This data is then used for multiple update *epochs*, in which the data is randomly shuffled and divided into mini batches, where stochastic gradient updates are performed to minimise the Actor and Critic loss functions separately, using standard Adam optimisers [16]–taking gradient descent steps on the loss functions with step sizes (or Learning Rate LR) α_a, for the actor and α_c for the critic, which we also anneal over iterations. This process is repeated over many iterations as the policy converges to the optimal policy π^*, which maximises the expected reward across episodes, or in our case, minimising key components of IS.

For stability, we also employ a pre-processing pipeline for raw observation vectors $s \in \mathcal{S}$. First, a logarithmic transformation, $\ln(1+x)$, is applied to specific features, $x \in s$, that were identified to be skewed, namely: normalised bid-side slippage, available spread, relative available bid-depth and the bid-ask spread. Following this we use a Running Mean Standard deviation (RMS) process on all observation features (except **Remaining Time**) to ensure inputs to the networks remain approximately zero-mean and unit-variance. We then clip the normalised values to a given range as to prevent outliers from destabilising the optimisation process.

A full treatment of the PPO algorithm can be found provided in Algorithm 2.

Algorithm 2 PPO Algorithm

initialise network parameters: θ_0, φ_0
for iterations i from $i = 1$ to I:
  anneal entropy scaling coeficient: $c_2 = c_2^{\text{start}} + (c_2^{\text{end}} - c_2^{\text{start}})(\frac{i-1}{I-1})$
  anneal actor LR: $\alpha_a^{'} = \alpha_a(1 - \frac{i-1}{I-1})$
  anneal critic LR: $\alpha_c^{'} = \alpha_c(1 - \frac{i-1}{I-1})$
  set $\theta_{\text{old}} \leftarrow \theta_i$
  collect batch of episodes $\mathcal{B}_i = \{\tau_1, \tau_2, ...\}$ under $\pi_{\theta_{\text{old}}}$
  compute returns $\hat{G}_t$ and advantage estimates $\hat{A}_t$ for all steps in $\mathcal{B}_i$ using GAE.
  process observations in $\mathcal{B}_i$: apply log transform, update RMS, normalise, and clip.
  standardise computed returns, $\hat{G}_t$, and Advantages $\hat{A}_t$ across $\mathcal{B}_i$
  for epochs j from $j = 1$ to E:
    shuffle collected data and create mini batches: $b \subset \mathcal{B}_i$
    for each mini batch b:
      update critic by descending the gradient of the critic MSE loss:
      $\varphi \leftarrow$ Adam step with $\alpha_c^{'}$ on $L(\varphi)$ using gradient norm clipping
      update actor by descending the gradient of the actor loss:
      $\theta \leftarrow$ Adam step with $\alpha_a^{'}$ on $L(\theta)$ using gradient norm clipping
    end for
  end for
end for

3.3.3 DRL Implementation Specifics

Here we outline some implementation specifics, including our specific network construction and hyperparameters presented in Table 3.1.

Network Architectures: The Actor and Critic networks are implemented using separate Multi-Layer Perceptrons (MLPs), with orthogonal weight initialisation and the following constructions:

- **Actor Network**: The Actor network maps the 9 dimensional observation vector to 13 logits corresponding to our action space, with three hidden layers of sizes 512, 256 and 128, each followed by ReLu activation. The 13 action logits are then passed through a softmax function producing a probability distribution.

- **Critic Network**: The Critic network maps the observation vector to a single scalar state-value estimate, with two hidden layers both of size 512, also using ReLu activations.

Table 3.1: Hyperparameters used in PPO

Hyperparameter	**Value**
Actor LR (α_a)	$1 \times 10^{-4} \rightarrow 0$
Critic LR (α_c)	$3 \times 10^{-4} \rightarrow 0$
Discount Factor (γ)	0.99
GAE Parameter (λ)	0.97
PPO Clip Range (ϵ)	0.18
Entropy Coef. (c_2)	$0.04 \rightarrow 0.01$
Epochs (E)	8
Mini-Batch Size	512 (steps)
Iterations (I)	8000
Observation Clip Range	$[-10, 10]$
Gradient Norm Clip Value	0.5

Chapter 4

Results & Analysis

Having established the methodological framework in the previous chapter–detailing the construction of the LOB simulation environment and the PPO-based DRL liquidation agent–this chapter now considers the experimental validation of this framework. In particular, we begin by validating the LOB simulator against several key statistics observable in the held out dataset. Following this, we deploy the agent within the environment and compare its liquidation performance against standard industry benchmarks.

4.1 K-NN LOB No-Trade Simulation Results

To validate the statistical fidelity of our simulator (Algorithm 1), we first consider the no-agent simulation behaviour, in which we compare several simulated distributions against the held-out empirical dataset using the two-sample Kolmogorov-Smirnov (KS) test, following the methodology in [4].

The KS test is a non-parametric method which determines if two independent samples are drawn from the same underlying probability distribution, in doing so, comparing the shape of an entire distribution for a given variable. For a sample of observations, $Z = \{z_1, z_2, \ldots, z_n\}$, the test operates on the one-dimensional Empirical Cumulative Distribution Function (ECDF), $F_n(z)$, which is the function that returns the proportion

of observations less than or equal to a value z:

$$F_n(z) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}_{z_i \leq z}$$

Then given ECDFs for the historical data, $F_{\text{hist}}(z)$, and a simulated sample, $F_{\text{sim}}(z)$, the test's D-statistic is defined as the maximum absolute difference between the two:

$$D = \sup_z |F_{\text{hist}}(z) - F_{\text{sim}}(z)| \in [0, 1]$$

thus, $D = 0$ corresponds to identical distributions.

To test distributional statistics, we randomly sample 1000 historical windows in the held-out set of length $[0, 1800]$ (matching our liquidation problem), in which we collect distributions for certain key metrics that define a LOBs shape and dynamics, including: The ask and bid volumes across the top 4 levels, the log mid-price returns over several horizons and the order book imbalance across the top 5 levels. To generate simulated distributions of these metrics, we randomly initialise Algorithm 1 to states within the held out set, and run simulations of the same length with action $a = 0\%$ for each step. Following [4], we also run a "naive" benchmark, in which we sample transitions uniformly at random over the build set. The results of this test can be found presented in Table 4.1.

The KS test results in Table 4.1 validate the simulator's ability to replicate many of the key statistical properties observable in the held-out dataset, where it consistently outperforms the naive random sampling, with the single exception of a marginal difference in the order book imbalance metric. This performance is particularly evidenced in the simulated distributions of conditional market dynamics, in which the low D-statistics for multi-horizon log returns confirm the simulator's ability to generate realistic price paths through its state-dependent conditioning–a feature that is unsurprisingly absent in the naive random sampling.

Table 4.1: Kolmogorov-Smirnov D-Statistic Test Values

Feature	**Simulator**	**naive**
$V^b_{t,1}$	**0.0218**	0.0289
$V^b_{t,2}$	**0.0344**	0.0537
$V^b_{t,3}$	**0.0449**	0.1321
$V^b_{t,4}$	**0.1262**	0.1667
$V^a_{t,1}$	**0.0350**	0.0450
$V^a_{t,2}$	**0.0510**	0.0826
$V^a_{t,3}$	**0.0805**	0.1718
$V^a_{t,4}$	**0.0871**	0.2012
$\text{OBI}^{(5)}$	0.0347	**0.0324**
$\rho^{(1)}_t$	**0.0520**	0.4919
$\rho^{(10)}_t$	**0.0741**	0.4813
$\rho^{(30)}_t$	**0.0679**	0.4731
$\rho^{(60)}_t$	**0.0642**	0.4653
$\rho^{(120)}_t$	**0.0598**	0.4546

Note: Lower D-statistic values correspond to better distributional similarity, and are indicated in bold.

We also observe that the random sampling is often sufficient for replicating the unconditional volume distributions at the top levels ($V_{t,1,2}$) of the book, though this feature is seen to progressively degrade at deeper levels. In contrast, our simulator maintains a much stronger correspondence with the empirical distributions across the deeper levels. This suggests that the structure of deeper liquidity levels is conditioned more acutely on the current market state.

4.2 Validation of Trade-Induced Market Impact

Beyond replicating passive market dynamics, a crucial test of the simulator is its ability to generate realistic counterfactual price impact in response to agent trades. A well established stylised fact of financial markets is that market impact scales as a concave function of the parent order size that generated it. In fact, this relationship has been

found to consistently follow that of a square root scaling relationship, given by [2]:

$$\mathcal{I}(X) = -\mathbb{E}[\Delta p^* | X] = Y \sigma_D \sqrt{\frac{X}{V_D}}$$

where, Δp^*, is the log mid-price change over the trading horizon, σ_D and V_D are the realised daily volatility and traded volume respectively, Y is a constant of proportionality, specific to the stock, and X is the total shares sold. Given the above relation, it can be seen that market impact is expected to be approximately (to the first order) independent of both the trading horizon length and the trading strategy, Π, that was employed.

Accurately modelling this non-linear relationship is essential, as a misspecified impact model would cause a DRL agent to learn a suboptimal policy.

To evaluate the simulators market impact fidelity, a static execution schedule (equal-sized child orders placed every 90 seconds) was deployed for five different parent order sizes ($X = 8000, 10000, 12000, 14000, 16000$). Simulations were initialised in the same initial held-out sate, and were ran 1000 times for each parent order size, the resulting mean mid-price depression was then measured against the final mean no-trade path, thereby isolating the impact from any underlying market trend. Figure 4.1 illustrates the outcome for the parent order size $X = 10000$.

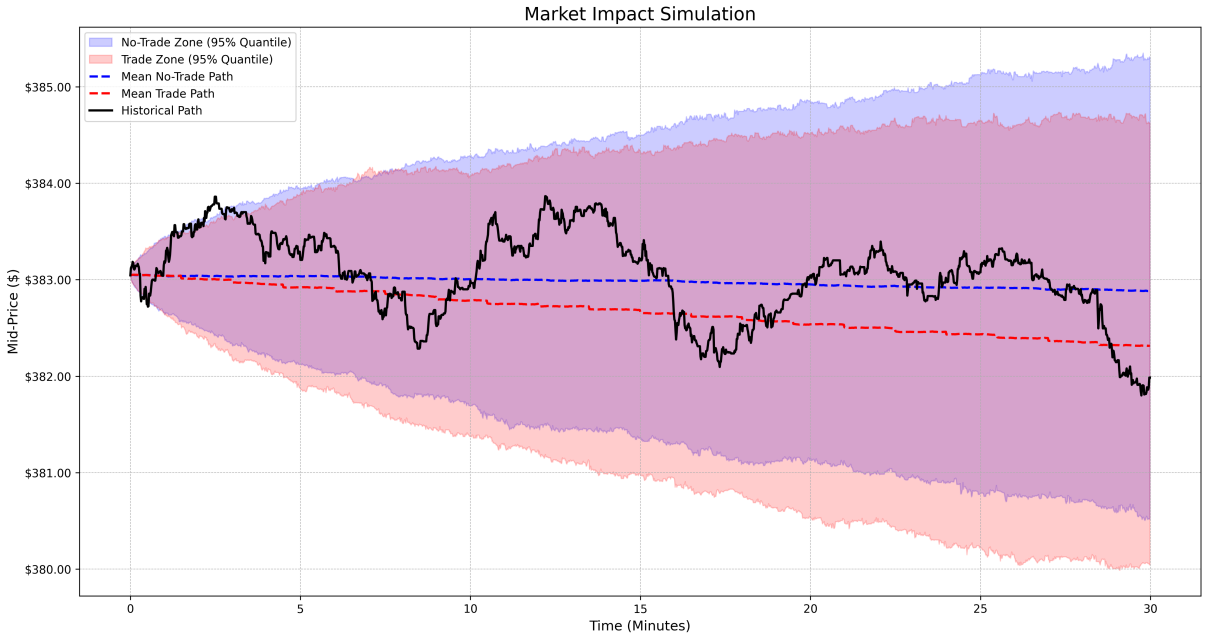

Figure 4.1: Market impact simulation, comparing the mean no-trade and trade price paths.

In Figure 4.1, we can see expected path simulation behaviour, with the historical path mostly contained within the 95% quantile of simulated paths. A clear price depression is also visible in the mean trade path relative to the no-trade path, with periodic price drops corresponding to the 90-second execution intervals. To quantify the impact scaling, we define the average impact, $\mathcal{I}(X')$, as the difference in the final log mid-price between the mean trade and no-trade paths, for an average executed volume of X' (accounting for variable available liquidity). A log-log linear regression was then preformed to estimate the scaling exponent α:

$$\ln(\mathcal{I}(X')) = \ln(C) + \alpha \ln(X')$$

for the following mean executed sizes: $X' = 7598, 9492, 11846, 13247, 15087$, and their corresponding mean impacts: $[0.001383, 0.0014270.001623, 0.001686, 0.001967]$.

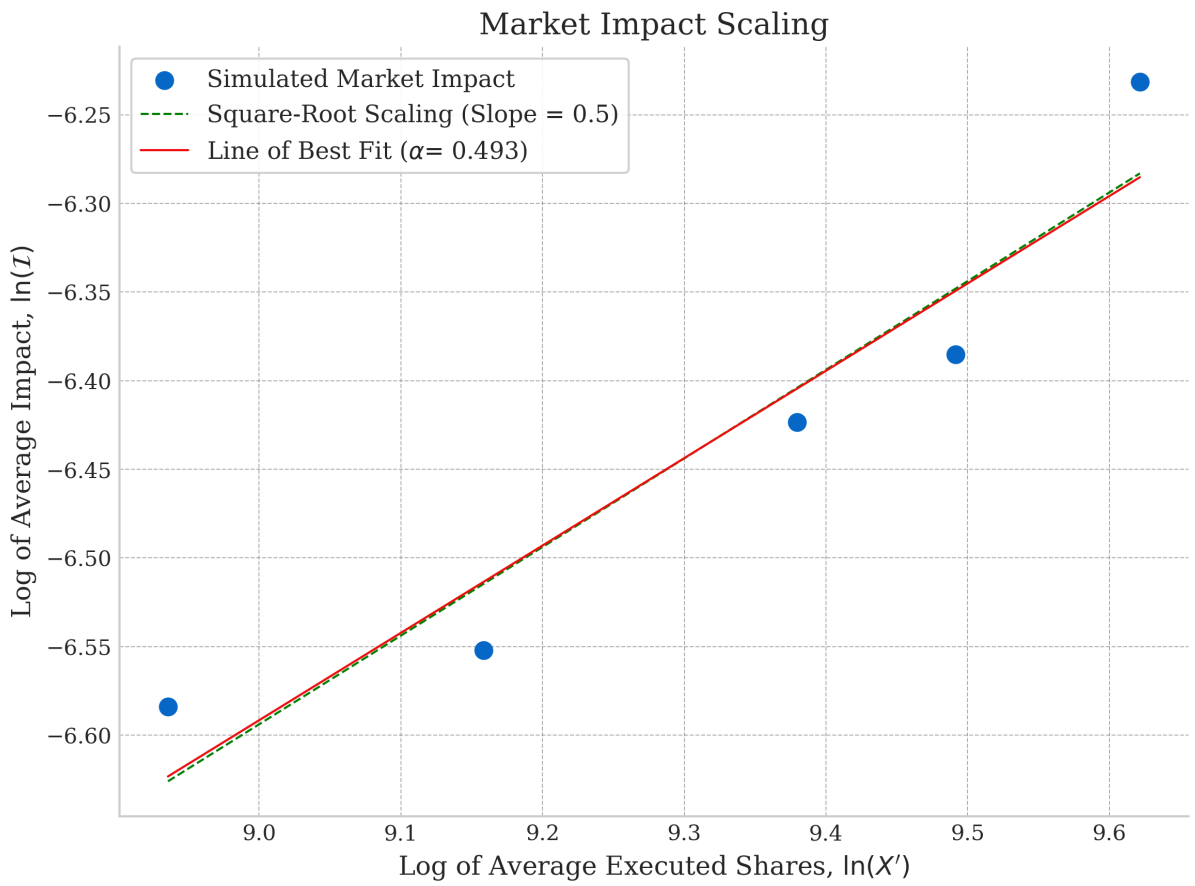

Figure 4.2: Log-log plot of simulated market impact versus executed share volume.

The regression yielded a scaling exponent of $\alpha = 0.493$ with an R^2 of 0.91, as shown in Figure 4.2. This result is in clear agreement with the empirically observed square-root scaling exponent of $\alpha \approx 0.5$, validating the simulators ability to implicitly and non-parametrically model realistic market impact induced by counterfactual agent trades.

4.3 DRL Liquidation Performance

Having validated the simulation environment, this section evaluates the performance of the trained DRL agent on the optimal liquidation task. Specifically, the agent was tasked with selling 20000 MSFT shares over a 30-minute (1800-seconds) horizon, with its performance benchmarked against the standard Time-Weighted Average Price (TWAP) strategy.

For a robust evaluation, we evaluate both strategies over 1000 episodes, each initialised in the held out set. Importantly, to ensure a fair comparison with TWAP, as to avoid remaining inventory held at T (which is included in the IS calculation), or large sells at $T-1$, we fix the TWAP schedule to sell 211 shares in the first step, with the remaining steps selling 11 shares (Totalling 20000)[1]. The primary performance metric employed was the mean and standard deviation of the IS, summarised in Table 4.2.

Table 4.2: Implementation shortfall results for both the DRL and TWAP implementations.

Strategy	**Mean IS**	**Std. Dev. of IS**
DRL Agent	\$6184.63	\$3282.81
TWAP	\$16290.37	\$14682.71

The DRL agent demonstrates a substantial improvement over the TWAP benchmark, across both the average cost and risk. As shown in Table 4.2, the agent achieves an average implementation shortfall of \$6184.63, a 62% reduction in average cost compared to TWAP. Even more critically, the agent exhibited a far superior reduction in implementation risk (IS variance), with its IS standard deviation being 77.6% lower than that of TWAP–a clear benefit from the risk aware agent.

To validate the statistical significance of these findings we perform two tests on the

[1]In our integer share environment a pure TWAP schedule would attempt to sell 11.11 shares at each step with only at most 11 being realised–thus leading to a last step sell of 211 shares, hence the fairest comparison is to sell the 211 shares on the first step with which variance risk is zero.

IS distributions. Specifically, a one-sided Welch's t-test was carried out to test the alternative hypothesis that the agent's mean IS is lower than that of TWAP. The test yielded a t-statistic of -21.24 (with $p \ll 0.001$), clearly rejecting the null, confirming statistical significance in mean IS reduction. Furthermore, a Levene's test for equality of variances produced a W-statistic of 1011.4 (with $p \ll 0.001$), again verifying the statistically significant reduction in risk. These findings are clearly demonstrated in the distributions of IS, illustrated in Figure 4.3.

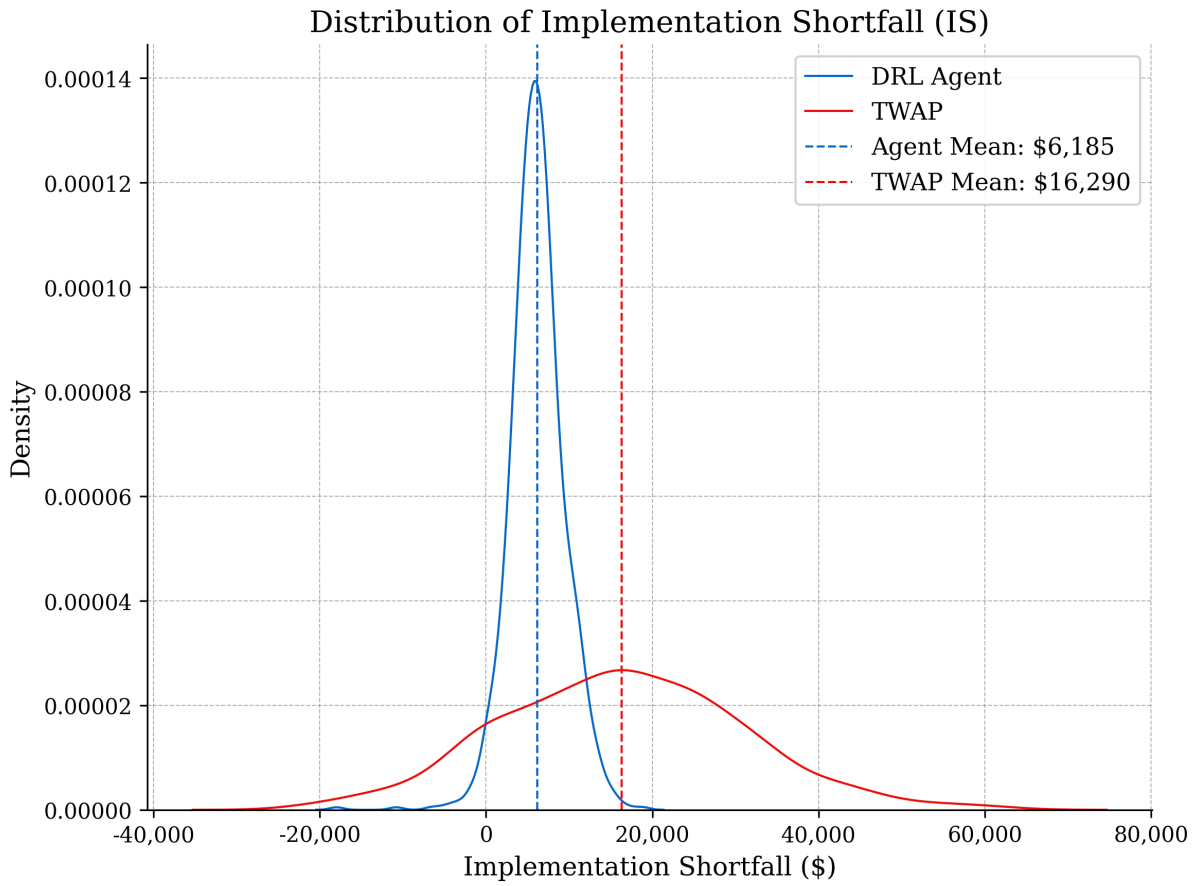

Figure 4.3: Comparative plot of Implementation Shortfall distributions for both the DRL agent and TWAP.

The distributions in Figure 4.3 provide a clear picture of the performance of both strategies. We find that the TWAP strategy had a 74.9% chance of producing an IS greater than the mean DRL IS. It can also be seen that both strategies occasionally receive negative IS (in which the revenue received was greater than the initial portfolio value), in particular, 2.3% of the time for the agent and 13.40% of the time for the TWAP schedule. The main point to note of course is the clear control of IS exhibited by the

agent, with an 85% quantile of \$9,380.65, in contrast to the risk exposed TWAP strategy with an 85% quantile of \$31215.79.

In Figure 4.4 we see the average sell trajectory produced by the agent across all simulations, along with the 95% quantile of realised paths taken. Clear similarities can be noted with average trajectory and that of the optimal paths under the Almgren & Chriss model, see Appendix 5. The average trajectory is front-loaded, consistent with risk-averse strategies that mitigate timing risk. However, the quantile zone demonstrates clear path dependence adaptability, likely explaining the agents strong control over IS variance within the stochastic environment.

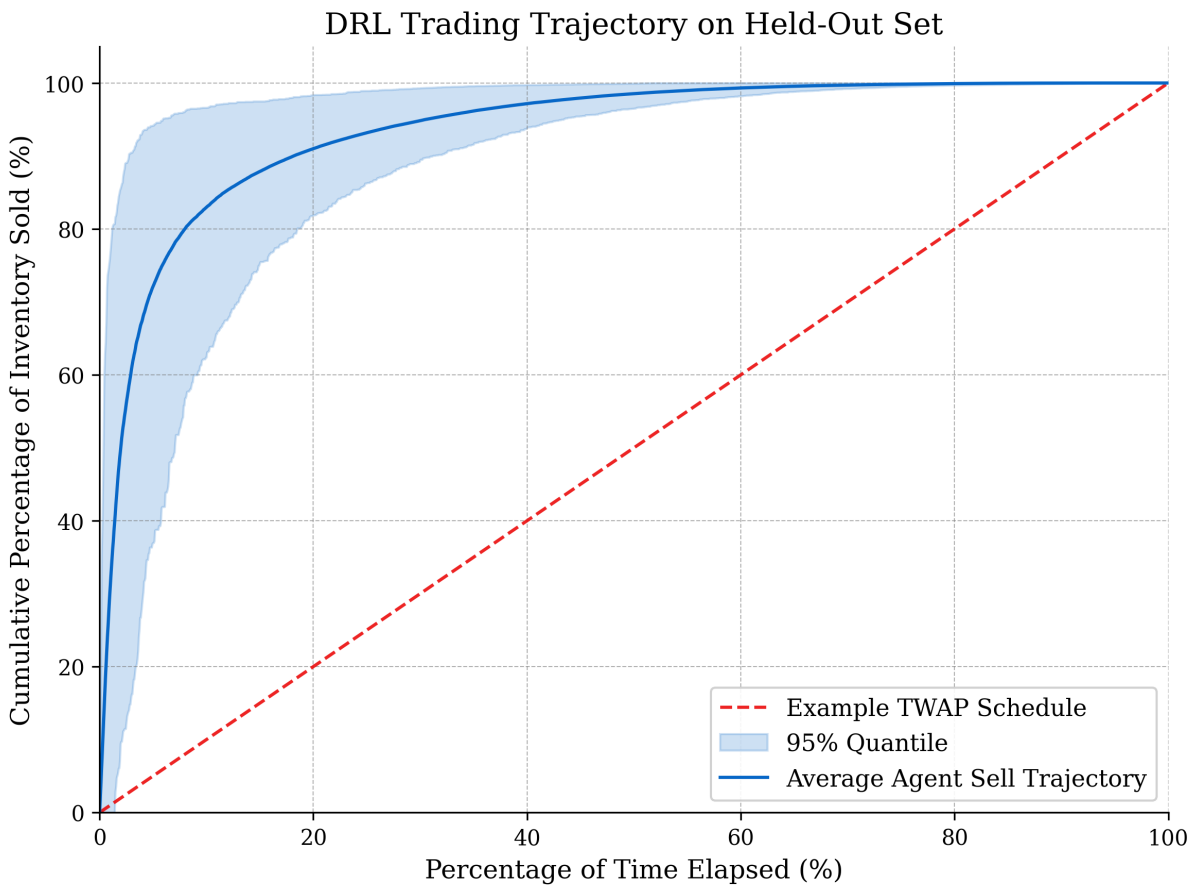

Figure 4.4: DRL agent trading trajectory over 1000 episodes initialised in the held out set.

We also note the agent successfully completed the liquidation task over every episode, with zero inventory remaining at T. The path dependence can be quantified by the following average seconds taken to sell a percentage of the initial inventory; steps to sell $25\% = 20.25 \pm 15.85$, steps to sell $50\% = 38.77 \pm 28.50$, steps to sell $90\% = 281.65 \pm 128.62$ and steps to sell $100\% = 1328.63 \pm 162.75$.

Chapter 5

Conclusion

To conclude, the work presented in this dissertation has considered a novel approach to both agent formulation and the non-parametric environment within which it was trained. At the core of our approach was the construction of a high-fidelity non-parametric Limit Order Book simulator using the K-Nearest Neighbour resampling method. Which served as the data-driven environment, in which we formulated and trained a PPO-based DRL agent for the task of optimal liquidation of MSFT shares.

The experimental evaluation of our trading simulator verified its ability to generate similar statistical distributions for key variables observable in the held-out data set, with KS D-statics in the range of $[0.052, 0.0679]$ across mid-price returns. Furthermore, we verified the simulators ability to correctly and implicitly model market impact, with an impact scaling exponent of $\alpha = 0.493$, in close accordance with the empirically observed value $\alpha \approx 0.5$.

In review of our trained DRL agent, we observed a strong comparative performance to that of the TWAP benchmark, with statistically significant reductions of 62% and 77.6% in both the mean and standard deviation of implementation shortfall respectively. Providing a compelling justification of our MDP formulation and the effectiveness of the risk proxy included in the agents reward function.

While these results are compelling, they should be considered alongside the inherent limitations of our approach. Namely our limited data set. Given the short period of market data, the regimes and market dynamics seen by the policy are undoubtedly a small subset of general market dynamics, and thus we cannot make any claims about the generality and robustness of the learnt policy. In addition to this several computational restrictions were accounted for which limit the "optimality" of the learnt policy, for example our restriction to a discrete action space. Ultimately, this research demonstrates significant potential for future development of DRL optimal execution agents within similar K-NN simulation environments.

Bibliography

[1] Robert Almgren and Neil Chriss. “Optimal execution of portfolio transactions”. In: *Journal of Risk* 3 (2001), pp. 5–40.

[2] Guillaume Maitrier et al. “The" double" square-root law: Evidence for the mechanical origin of market impact using Tokyo Stock Exchange data”. In: *arXiv preprint arXiv:2502.16246* (2025).

[3] Konark Jain et al. “Limit order book simulations: A review”. In: *arXiv preprint arXiv:2402.17359* (2024).

[4] Michael Giegrich, Roel Oomen, and Christoph Reisinger. “Limit Order Book Simulation and Trade Evaluation with K-Nearest-Neighbor Resampling”. In: *arXiv preprint arXiv:2409.06514* (2024).

[5] John Schulman et al. “Proximal policy optimization algorithms”. In: *arXiv preprint arXiv:1707.06347* (2017).

[6] Anna A Obizhaeva and Jiang Wang. “Optimal trading strategy and supply/demand dynamics”. In: *Journal of Financial markets* 16.1 (2013), pp. 1–32.

[7] Andrew W Lo and A Craig MacKinlay. “Stock market prices do not follow random walks: Evidence from a simple specification test”. In: *The review of financial studies* 1.1 (1988), pp. 41–66.

[8] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. second. The MIT Press, 2018.

[9] Yadh Hafsi and Edoardo Vittori. “Optimal execution with reinforcement learning”. In: *arXiv preprint arXiv:2411.06389* (2024).

[10] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. In: *nature* 518.7540 (2015), pp. 529–533.

[11] Siyu Lin and Peter A Beling. “An end-to-end optimal trade execution framework based on proximal policy optimization”. In: *Proceedings of the twenty-ninth international conference on international joint conferences on artificial intelligence*. 2021, pp. 4548–4554.

[12] Alessandro Micheli and Mélodie Monod. “Deep Reinforcement Learning for Online Optimal Execution Strategies”. In: *arXiv preprint arXiv:2410.13493* (2024).

[13] Databento. *MSFT Limit Order Book Data (XNAS.ITCH, schema mbp-10)*. https://databento.com. Accessed: 2025-03-23, covering 10 February 2025 to 18 March 2025. 2025.

[14] Alvaro Cartea, Ryan Donnelly, and Sebastian Jaimungal. “Enhancing trading strategies with order book signals”. In: *Applied Mathematical Finance* 25.1 (2018), pp. 1–35.

[15] John Schulman et al. “High-dimensional continuous control using generalized advantage estimation”. In: *arXiv preprint arXiv:1506.02438* (2015).

[16] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *arXiv preprint arXiv:1412.6980* (2017).

Appendix

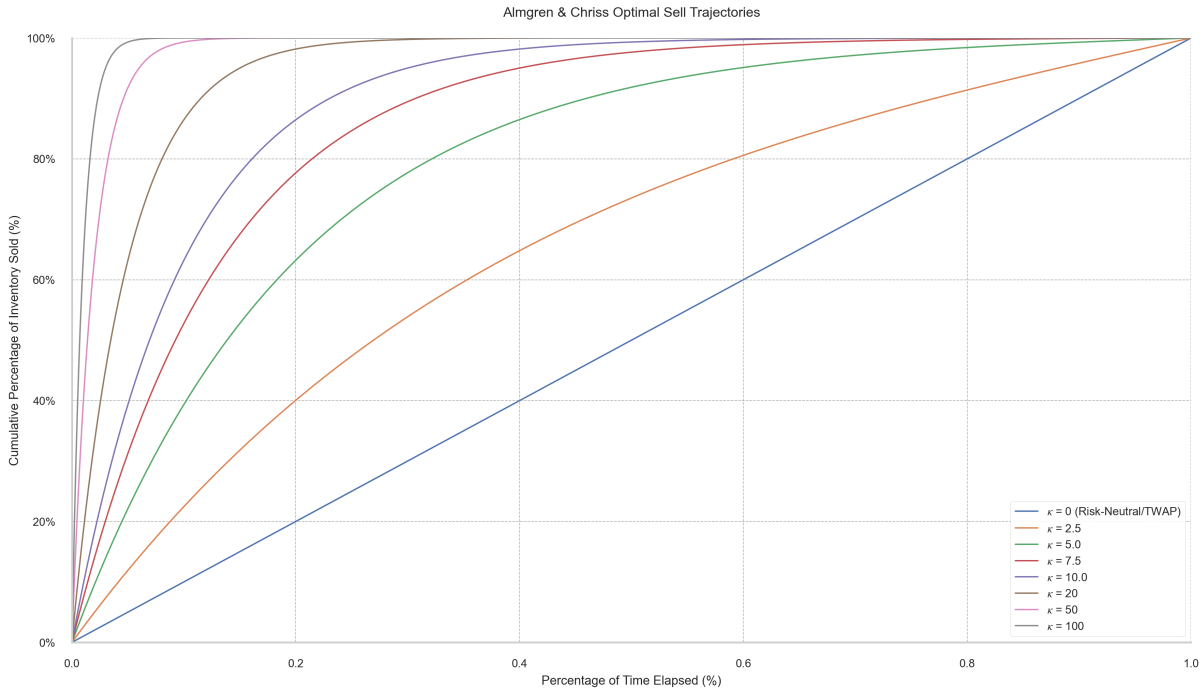

Figure 1: The Efficient Frontier of globally optimal static strategies under the Almgren & Chriss model.