LUISS

Department of Economics and Finance

Degree Program in Economics and Finance major in Finance

OPTIMAL EXECUTION USING REINFORCEMENT LEARNING

Supervisor:
Prof. Nicola Borri
Co - Supervisor:
Prof. Megha Patnaik

Candidate:
Andrea Quaranta
ID: 778071

Academic Year 2024/2025

Ai miei genitori,
a mia zia,
a Dada,
e a Gabriella.

Contents

Chapter 1

Introduction

Optimal execution (OE) has been a central topic in the financial industry and has been extensively studied by both practitioners and academics due to its significant impact on the profitability of trading strategies and on the investment decisions of banks, hedge funds, proprietary trading firms, and asset management companies. Transaction costs, in particular, are widely recognized as a major determinant of investment performance (Freyre-Sanders et al., 2004), directly influencing the realized returns of active strategies and the liquidity profile of institutional portfolios. The rise of algorithmic trading and electronic markets has further increased the need for adaptive and dynamic execution strategies that can respond to real-time market conditions. The literature highlights that a single large order can consume all available liquidity at the best bid or ask price, since the immediate depth of the order book is limited (Kyle, 1985). To mitigate market impact, practitioners typically split large orders into smaller increments executed over time. This introduces a trade-off: rapid execution reduces risk exposure but tends to increase price impact, while slow execution lowers immediate impact but exposes the trader to price volatility and timing risk (Obizhaeva and Wang, 2013; Almgren and Chriss, 2001). Traditional execution algorithms such as Time-Weighted Average Price (TWAP) and Volume-Weighted Average Price (VWAP) attempt to systematically slice orders but often fail to adapt to evolving market dynamics, leading to suboptimal performance (Kissell and Malamut, 2006). Recent studies have explored reinforcement learning approaches to address these shortcomings by enabling the learning of adaptive policies that react to stochastic and high-frequency market conditions (Nevmyvaka et al., 2006; Spooner et al., 2018). Building upon the foundational market microstructure models by Kyle (Kyle, 1985), Glosten and Milgrom (Glosten and Milgrom, 1985), and the optimal execution framework by Almgren and Chriss (Almgren and Chriss, 2001), this thesis investigates whether a model-free reinforcement learning (RL) method—specifically a Deep Q-Network (DQN) (Mnih et al., 2015)—can effectively optimize the execution of large block orders within a constrained timeframe. The agent is trained in an Agent-Based Interactive Discrete Event Simulation (ABIDES) environment (Byrd et al., 2019) and its Gym wrapper extension (Amrouni et al., 2021), with the objective of minimizing transaction costs arising from market impact and bid-ask spreads, while adapting dynamically to the simulated order book state. This work contributes to the literature by integrating classical market microstructure theory with modern reinforcement learning techniques, demonstrating how simulation-based approaches can enhance trading strategy design under realistic and complex market dynamics.

1.1 Research Objectives

This thesis contributes to the literature on optimal execution by combining reinforcement learning techniques with a realistic simulation-based trading environment. The work is structured around five main objectives:

1. Leverage the ABIDES-Gym framework to train and evaluate a Deep Q-Network (DQN) agent within a discrete-event simulation of a limit order book (LOB), replicating microstructural features of modern electronic markets.

2. Define an extended discrete action space that allows the agent to choose among multiple order types (market, limit), order sizes, and the option to remain inactive, thereby enabling adaptive control over execution aggressiveness throughout the episode.

3. Design and implement a parametric reward function that integrates core execution objectives. The function provides dense feedback at each step, balancing the penalty from implementation shortfall with a risk-aversion term proportional to the rolling variance of execution costs.

4. Apply systematic hyperparameter optimization using Optuna, focusing on key parameters such as learning rate, discount factor, batch size, network architecture, and risk aversion.

5. Benchmark the trained DQN agent against classical execution strategies (TWAP, aggresive, random), evaluating performance in terms of transaction costs, slippage, and robustness under stochastic market conditions.

Through this integration of market microstructure modeling, agent-based simulation, and deep reinforcement learning, the thesis investigates whether data-driven execution policies can outperform static heuristic benchmarks under realistic trading conditions.

Chapter 2

Background and Literature Review

2.1 Market Microstructure

Market microstructure investigates the set of rules, mechanisms, and behaviors that define trading and price formation in financial markets. Unlike frictionless pricing models, microstructure theory closely examines how trading protocols and information flows affect transaction costs, liquidity, price discovery, and volatility (O'Hara, 1995). Foundational studies such as those by Kyle (1985), Glosten and Milgrom (1985), and O'Hara (1995) form the backbone of this literature. Kyle (1985) introduced the influential concept of market impact: how private information and the execution of sizable orders moves both prices and observed liquidity. Crucially, market impact tends to be proportional to order size, and its effects are more pronounced in illiquid environments. This link between order flow and price formation marks a major departure from classical, frictionless models. The model by Glosten and Milgrom (1985) focuses on information asymmetry between participants. Their analysis demonstrates why a positive bid-ask spread emerges—even in competitive markets—as a necessary compensation for liquidity providers facing the risk of adverse selection from informed traders. These elements—spread, finite depth, imbalance, and market impact—are what make trading a quantitative challenge and motivate research in optimal execution. For practitioners and researchers designing optimal execution (OE) strategies, microstructure is essential. In an idealized, frictionless market as in Black–Scholes, large trades could be carried out instantly without cost or impact. Microstructure theory, by contrast, shows that spreads, limited depth, order book imbalance and market impact are present, making the execution of large orders both costly and risky (Kyle, 1985; Glosten and Milgrom, 1985). The choice of state variables in OE models—such as spread, imbalance, mid-price, and book depth—stems directly from microstructure insights. For example, in the framework of Glosten and Milgrom (1985), the spread measures informational cost and becomes naturally embedded as a penalizing term in any execution strategy. Similarly, the notion of linear market impact introduced by Kyle (1985) justifies why the OE policy must actively minimize the trading footprint. These theoretical links are the methodological basis of the models considered in this thesis. Altogether, these considerations motivate a focused study of the limit order book (LOB), the principal microstructural object in modern electronic markets.

2.2 Limit Order Book

The limit order book (LOB) is the fundamental electronic registry in modern financial markets, recording all outstanding buy and sell orders at various price levels. Each order is characterized by a limit price and quantity, and the structure of the book evolves dynamically as traders place, cancel, or execute orders. Orders in the LOB are commonly prioritized by price—where better prices for buyers or sellers receive priority—and, within each price level, by arrival time, following the principle of price–time priority. The trading mechanism operates in discrete increments, known as the *tick size*, which determines the smallest permissible difference between prices in the book.

2.2.1 Key Variables

Several key variables emerging from the LOB play a central role in the analysis of trading and the formulation of optimal execution strategies:

- **Total depth:** cumulative volume across the first k price levels (d total) on side h:

$$TD_h^k = \sum_{j=1}^{k} Q_h^j, \qquad k \in \{1, \dots, d\},\ h \in \{\text{bid}, \text{ask}\},$$

where Q^j_{ask} (Q^j_{bid}) is the outstanding limit-order volume at the j-th best ask (bid) level.

- **Volume imbalance:** relative difference between bid- and ask-side depth up to level k:

$$v^k = \frac{TD^k_{\text{bid}} - TD^k_{\text{ask}}}{TD^k_{\text{bid}} + TD^k_{\text{ask}}}, \qquad k \in \{1, \dots, d\}.$$

- **Mid-price:** midpoint between best bid and best ask:

$$P_{\text{mid}} = \frac{P_{\text{best ask}} + P_{\text{best bid}}}{2}.$$

- **Spread:** difference between best ask and best bid:

$$\text{Spread} = P_{\text{best ask}} - P_{\text{best bid}}.$$

2.2.2 Order Types

Trading on a limit order book means choosing among several types of orders, each with specific effects on execution speed and certainty. The limit order allows traders to specify both price and quantity; it is added to the book and only executed if the market reaches the desired price. Execution is not guaranteed, and among orders at the same price level, those placed earlier are matched first according to price–time priority, often referred to as First-In-First-Out (FIFO). Market orders take a different approach: they execute instantly against the best available price—buys match with existing sell orders starting from the lowest ask, sells match with buy orders starting from the highest bid. Market orders ensure prompt execution but the final price may be less favorable, especially when liquidity is low or spreads are wide. There are also specialized order types like fill-or-kill, which require complete and immediate execution or cancellation, and immediate-or-cancel, which try to fill as much as possible instantly. These mechanisms underpin the practical trading strategies that interact directly with spread, depth, and book dynamics. These order types capture the basic trade-off between speed, cost, and certainty, which stands at the core of the optimal execution problem.

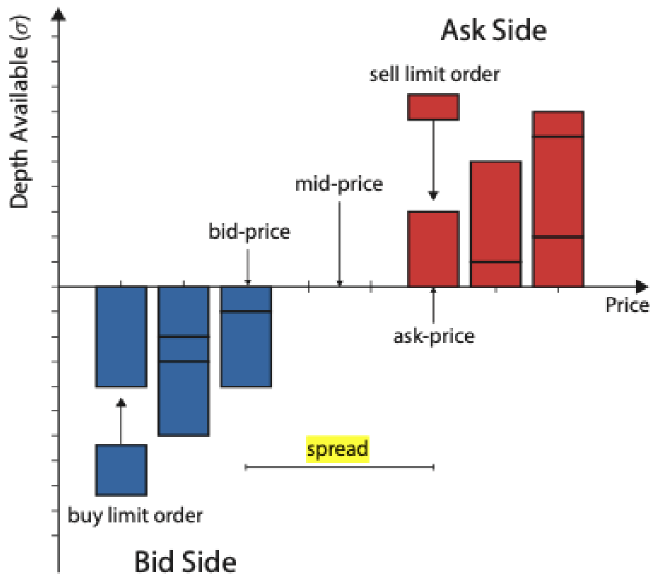

Figure 2.1: Schematic representation of a limit order book (LOB). Adapted from Gould et al. (2013).

2.2.3 Literature on the LOB

Theoretical understanding of the LOB dates back to early contributions. Stigler (1964) introduced a stochastic framework to model the evolution of order books, and Garman (1976) developed models to explain price formation under order-driven protocols. Modern stochastic approaches are exemplified by Cont et al. (2010), who model LOB dynamics using Poisson processes for order arrivals and cancellations, enabling analytic investigation of market liquidity and execution probabilities. A detailed empirical and theoretical survey is provided by Gould et al. (2013), which discusses stylized facts, variable definitions, and unresolved questions in the study of LOBs. Additionally, Parlour and Seppi (2008) analyze both theoretical and empirical aspects of limit order markets, including strategic behavior and market design. The connection between LOB theory and optimal execution is direct: execution strategies are carried out within the LOB, with agents choosing between aggressive (market orders) and passive (limit orders) tactics based on book conditions. Central variables like spread and imbalance serve as immediate signals for reinforcement learning policies and stochastic control algorithms. The selection and timing of trades thereby depend crucially on the real-time structure and dynamics of the LOB, highlighting the interplay between empirical observables and theoretical constructs in quantitative finance.

2.3 Optimal Execution

2.3.1 Problem Formulation

The optimal execution task can be described as the problem of liquidating or acquiring a parent order of size X_0 over a predetermined horizon T. Evaluating such a strategy requires taking into account multiple sources of cost: the immediate price impact generated by submitting orders, the short-term volatility that may move prices during the execution, and possible longer-term distortions in supply and demand that influence future prices. In addition, a penalty is typically introduced whenever the order is not fully executed within the time window $[0, T]$. Without loss of generality, we focus on the problem of liquidating an initial position X_0. In a discrete-time setting with $N + 1$ periods, the execution strategy is modeled as a sequential decision process. At each time step $t_k = \frac{k}{N}T$ for $k \in \{0, \ldots, N\}$, the trader decides on the quantity of shares to execute. Let x_k denote the remaining inventory at time t_k, with the process beginning at an initial inventory of $x_0 = X_0$ and ending with a final inventory of $x_N = 0$. The quantity of shares executed between t_{k-1} and t_k is denoted by u_k, where $u_k = x_{k-1} - x_k$ and $u_k \geq 0$. Full liquidation is guaranteed by the terminal condition $x_N = 0$, which implies that the total number of shares liquidated equals the initial inventory:

$$\sum_{k=1}^{N} u_k = \sum_{k=1}^{N} (x_{k-1} - x_k) = (x_0 - x_1) + (x_1 - x_2) + \cdots + (x_{N-1} - x_N) = x_0 - x_N = X_0.$$

Let P_k be the execution price at time t_k. The total cost of liquidation is the sum of the transaction costs at each step:

$$C = \sum_{k=1}^{N} P_k u_k.$$

The objective of the optimization is to find the sequence of executed quantities $\{u_1, \dots, u_N\}$ that minimizes the expected total cost:

$$\min_{\{u_1,\dots,u_N\}\in\mathbb{R}_+^N} \mathbb{E}[C].$$

2.3.2 Classical Models

Building on this formal setup, the earliest rigorous treatment was given by Bertsimas and Lo (1998), who introduced a dynamic programming approach. Their framework captures the trade-off between immediate execution, which incurs higher market impact, and slower execution, which reduces impact but increases exposure to price volatility. In certain specifications, closed-form solutions for optimal trading trajectories can be derived, providing rigorous guidance for execution strategies. A major step forward was made by Almgren and Chriss (2001), whose model remains the standard benchmark in the literature. They decomposed market impact into temporary and permanent components, modeling the transaction price as

$$P_k = S_k + \eta x_k + \gamma X_k,$$

where S_k is the unaffected price, η is the temporary impact per share, and γ the permanent impact coefficient. The objective is to minimize expected implementation shortfall:

$$\text{IS} = \mathbb{E}\left[\sum_{k=0}^{N}(P_k - S_0)x_k\right],$$

while also controlling for risk via the variance of IS. This leads to a quadratic optimization problem:

$$\min_x \mathbb{E}[\text{IS}] + \lambda \operatorname{Var}[\text{IS}],$$

where λ is a risk-aversion parameter. Varying λ traces out the well-known risk–cost efficient frontier: aggressive strategies achieve faster liquidation but incur larger impact, while conservative ones reduce impact at the cost of higher risk. Subsequent research refined these classical frameworks. Huberman and Stanzl (2004) provided conditions that rule out price manipulation, ensuring that no strategy can generate systematic arbitrage profits from impact models. Guéant (2016) advanced optimal execution theory by incorporating more realistic microstructural details, such as discrete pricing, random order arrival, and operational constraints, yielding strategies that are closer to practice. Finally, Obizhaeva and Wang (2013) emphasized the role of limit order book resilience, modeling how quickly liquidity replenishes after trades. Their results highlight that slow resilience amplifies market impact, leading to more cautious execution paths and a deeper understanding of true costs in modern markets. Together, these contributions form the foundation of optimal execution theory: from the dynamic programming perspective of Bertsimas–Lo, to the risk–impact frontier of Almgren–Chriss, and subsequent refinements ensuring robustness, realism, and microstructural consistency.

2.4 Optimal Execution: Reinforcement Learning Approaches

The early models of Bertsimas and Lo (1998) and Almgren and Chriss (2001) formed the basis of optimal execution research and are still regarded as the standard starting point in the field. Despite their analytical elegance, these frameworks rely on restrictive assumptions: linear market impact, absence of rich market information, and closed-form strategies with limited adaptivity to real-world complexity. As financial markets have evolved, they have become markedly more non-linear, information-rich, and structurally complex, with electronic trading generating vast amounts of granular data. These developments challenge the adequacy of static analytic solutions and motivate a shift toward algorithms that learn directly from data or high-fidelity simulation. Reinforcement learning (RL) helps address these challenges by letting agents improve their execution strategies through interaction with market data or simulated environments. RL methods learn policies that adapt to changing market conditions, capturing dependencies and patterns that static models often overlook. The first major application of RL to optimal execution was proposed by Nevmyvaka et al. (2006), who used RL to minimize implementation shortfall in realistic trading scenarios. This early work demonstrated how RL agents can outperform benchmark algorithms in dynamic environments. Subsequent advances include hybrid frameworks such as Hendricks and Wilcox (2014), who combined Q-learning with the Almgren–Chriss trajectory to exploit state-dependent trade scheduling. Ning and Zhang (2018) introduced Deep Q-Networks (DQN), which combine deep neural networks with Q-learning. This approach made it possible to handle high-dimensional features and reduce the curse of dimensionality in execution problems, building on the deep learning framework of Mnih et al. (2015). Modern RL research for optimal execution frequently deploys deep reinforcement learning, with works by Lin and Beling (2020), Wei et al. (2019), and Spooner et al. (2018) applying algorithms such as Proximal Policy Optimization (PPO; Schulman and Klimov, 2017) for improved adaptivity. These methods incorporate temporal correlations and leverage limit order book (LOB) data, but face the ongoing difficulty of realistically modeling agent-driven market impact when relying solely on historical datasets. To address this challenge, recent studies have explored multi-agent reinforcement learning approaches, often relying on dedicated market simulators to recreate realistic trading environments. In these settings, RL agents learn execution strategies by interacting with heterogeneous participants, rather than passively replaying historical data. This line of research is developed further in the next section, where we discuss the role of agent-based models and ABIDES in particular.

2.4.1 Limitations of Historical Data and Agent-Based Simulation

Traditional backtesting with historical market data remains the baseline for evaluating trading strategies, but its limitations are well-documented. Historical approaches simply replay observed transactions; they cannot account for the experimental agent's influence on prices or liquidity, nor replicate unpredictable or rare scenarios. Because markets in backtests are unresponsive, results may be misleading, especially for adaptive or reinforcement learning-based execution algorithms intended for real environments. Agent-based models (ABMs) address these shortcomings by simulating markets as systems of multiple autonomous agents, each governed by its own strategy or behavioral rules. This framework reflects the diversity of real traders and allows realistic supply–demand dynamics and emergent patterns that static, single-agent approaches cannot reproduce. These models occupy a middle ground between perfect rationality and zero-intelligence, allowing agent rules to be empirical or theoretical. Important contributions to limit order book modeling include the works of Alfi et al. (2009), Chakraborti et al. (2011), Hamill and Gilbert (2015), and Coletta et al. (2022). Among the available simulators, ABIDES stands out as a flexible, high-fidelity framework for electronic market design. It supports customizable populations of agents—such as market makers, momentum traders, value investors, and noise traders—and optionally incorporates a reference or fundamental value process, often modeled with an Ornstein–Uhlenbeck (OU) dynamic. This OU process acts as a mean-reverting signal or latent value that agents may use in their strategies, but crucially, the actual transaction prices and limit order book states emerge from agent interaction rather than directly following the OU process. Applications to optimal execution demonstrate the value of this architecture: reinforcement learning agents interact with heterogeneous participants in ABIDES, and strategies can be benchmarked under realistic market impact, learning from feedback and emergent behaviors. Recent works such as Karpe and Roux (2020), Nagy et al. (2023), and Hafsi and Vittori (2024) apply reinforcement learning to optimal execution within agent-based simulated markets, focusing on dynamic interaction and market adaptation. As shown in the figure below, adapted from Coletta et al. (2022), the core of our approach is to move beyond static historical backtesting (A). Instead, we use a multi-agent simulation (B) where the experimental agent competes with and adapts to a diverse population of market participants, creating a truly responsive environment.

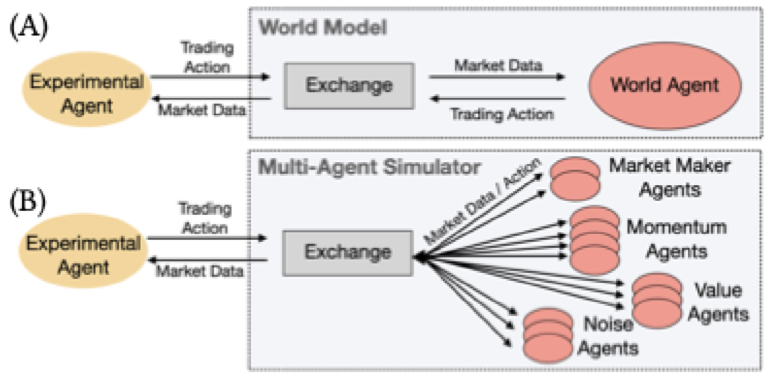

Figure 2.2: Comparison of agent evaluation methods. Panel (A) shows static historical backtesting, while panel (B) illustrates a multi-agent simulation (ABIDES) where the experimental agent interacts with a responsive market. Adapted from Coletta et al. (2022).

Chapter 3

Simulation Environment and Methodology

3.1 Reinforcement Learning Framework

3.1.1 Overview of Reinforcement Learning

Reinforcement learning (RL) is a foundational branch of machine learning, distinct from both supervised and unsupervised learning. In supervised learning, models are trained using labeled data to predict outputs for new samples, relying on explicit feedback for correctness. Unsupervised learning, on the other hand, aims to discover hidden structures in unlabeled data, such as clusters or latent factors, without specific guidance or targets. RL departs from these paradigms by focusing on sequential decision-making in an interactive environment (see Fig. 3.1).

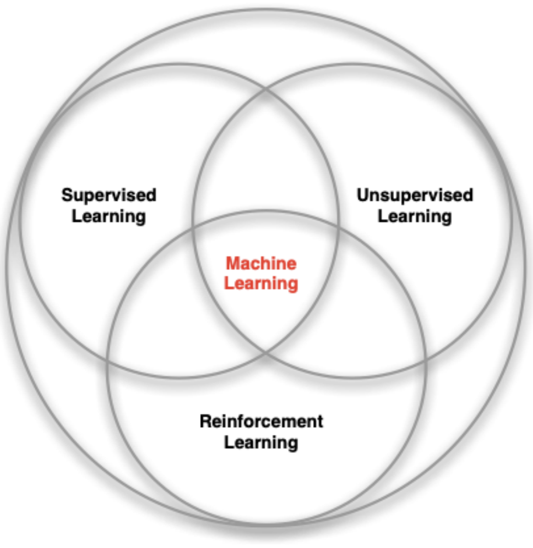

Figure 3.1: Supervised, unsupervised, and reinforcement learning as the three main branches of machine learning.

There is no external supervisor providing correct answers; instead, an agent explores a space of possible actions and receives rewards or penalties as feedback, often delayed and sparse. Through repeated interaction, the agent learns a policy—a mapping from observed states to actions—that aims to maximize cumulative rewards over time. The classical RL framework consists of an agent, an environment, state and action spaces, a reward function, and a policy. At each time step, the agent observes the current state, chooses an action, and receives both a new state and a reward determined by the environment's dynamics (see Fig. 3.2). The learning process involves balancing exploration (testing new actions to discover their effects) and exploitation (leveraging learned strategies with known outcomes).

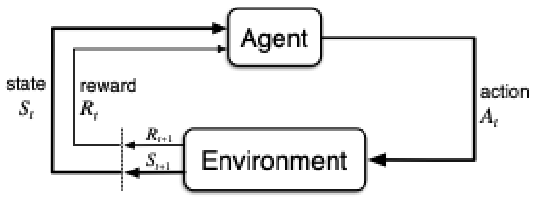

Figure 3.2: Standard agent–environment interaction loop in reinforcement learning. The agent selects an action A_t based on the current state S_t, receives a reward R_{t+1}, and transitions to a new state S_{t+1}.

In finance, RL methods have proven especially powerful in domains like algorithmic trading, portfolio optimization, and optimal execution, where environments are stochastic, data is sequential, and traditional rule-based models often fail to adapt to rapidly changing market conditions. By learning directly from experience, RL agents can dynamically adjust their strategies in real time, aiming to optimize objectives such as risk-adjusted returns or minimization of transaction costs.

3.1.2 Markov Decision Processes (MDP)

To formalize reinforcement learning, the interaction between agent and environment is typically modeled as a Markov Decision Process (MDP). An MDP is defined as a tuple $\langle S, A, P, R, \gamma, \mu \rangle$, where S is the set of states, A the action space, $P(s'|s,a)$ the Markovian transition probability, $R(s,a)$ the reward function, $\gamma \in [0,1)$ the discount factor, and μ the initial state distribution. A policy $\pi(a|s)$ specifies the probability of taking action a in state s. The agent's objective is to maximize the expected return

$$J(\pi) = \mathbb{E}_\pi \left[\sum_{t=0}^{T} \gamma^t r_t \right],$$

that is, the discounted cumulative reward over a finite horizon T. This mathematical formalism provides the basis for defining execution as a sequential decision problem: at each time step, the agent decides how much and how aggressively to trade, receiving feedback in the form of transaction costs and execution risk.

3.1.3 From MDP to Q-learning

A central concept in RL is the state–action value function, or Q-function,

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{T}\gamma^{t} r_t \;\middle|\; s_0 = s, a_0 = a\right],$$

which quantifies the expected return starting from state s and action a under policy π. Tabular Q-learning algorithms iteratively update estimates of $Q(s,a)$ to approximate the optimal value function $Q^{*}(s,a)$. However, in complex environments such as limit order books, the state space is extremely high-dimensional, making tabular methods infeasible. This motivates the use of function approximation techniques to generalize across states and actions.

3.1.4 Deep Q-Network (DQN)

The Deep Q-Network (DQN; Mnih et al., 2015) addresses this challenge by using a neural network parameterized by θ to approximate the value function, $Q(s,a;\theta)$. The algorithm stabilizes learning with two key mechanisms: (i) an experience replay buffer that stores transitions (s,a,r,s') and enables minibatch updates, and (ii) a target network that provides more stable Q-value targets during training. The loss minimized at each step is

$$L(\theta) = \mathbb{E}_{(s,a,r,s')}\Big[\big(r + \gamma \max_{a'} Q(s',a';\theta^{-}) - Q(s,a;\theta)\big)^{2}\Big],$$

where θ^{-} denotes the parameters of the target network. This framework allows the agent to approximate optimal decision policies even in environments with high-dimensional and noisy inputs, such as simulated order books.

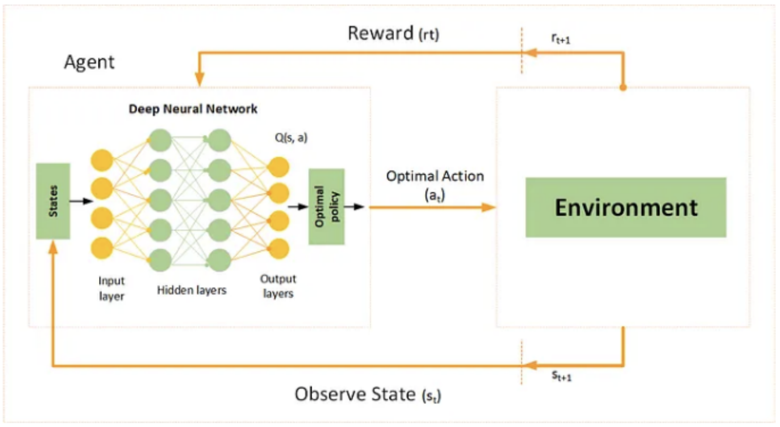

Figure 3.3: Schematic representation of the Deep Q-Network (DQN). The agent uses a neural network to approximate $Q(s, a; \theta)$ and selects actions to maximize expected reward based on observed states.

3.1.5 Extensions of DQN

Several extensions improve the robustness of the original DQN algorithm. Double Q-learning mitigates the overestimation bias of action values. Dueling networks separate the estimation of state value and action advantage, improving stability in environments with many similar-valued actions. Prioritized experience replay improves sample efficiency by giving higher probability to transitions with larger temporal-difference errors. These refinements are particularly relevant in financial markets, where noise, sparsity of rewards, and sensitivity to execution timing demand stable and efficient learning.

3.1.6 Why DQN for Optimal Execution

The choice of DQN in this thesis is motivated by the discrete nature of the action space in optimal execution (market orders, limit orders, or hold, with different sizes), which aligns directly with the Q-learning framework. Alternative policy-gradient algorithms such as PPO or SAC are better suited for continuous action spaces but less efficient for discrete order execution. Moreover, DQN is model-free, requiring no assumptions about the underlying market dynamics, and has been successfully applied to optimal execution in prior studies (Nevmyvaka et al., 2006; Hafsi and Vittori, 2024). By leveraging DQN together with its extensions, the agent is expected to learn adaptive strategies that outperform static benchmarks under realistic market microstructure conditions.

3.2 MDP Representation of the Execution Problem

3.2.1 State representation

The experimental agent interacts with the market through a structured state representation, inspired by the ABIDES-Gym framework (Byrd et al., 2020) and adapted to include execution-specific features. Formally, the state at time t is defined as

$$\begin{aligned} s(t) = \big(&holdingsPct_t,\, timePct_t,\, differencePct_t,\, imbalance5_t, \\ &imbalanceAll_t,\, priceImpact_t,\, spread_t,\, directionFeature_t,\, R_t^k\big). \end{aligned} \tag{3.1}$$

where:

- $holdingsPct_t = \dfrac{holdings_t}{parentOrderSize}$: execution progress as fraction of the parent order,
- $timePct_t = \dfrac{t - startingTime}{timeWindow}$: elapsed time within the execution horizon,

- $differencePct_t = holdingsPct_t - timePct_t$: progress relative to uniform schedule,
- $priceImpact_t = midPrice_t - entryPrice$: realized price impact,
- $imbalanceAll_t = \dfrac{\text{bid volume}}{\text{bid volume} + \text{ask volume}}$: order book imbalance across all levels,
- $imbalance5_t = \dfrac{\text{bid volume (top-5)}}{\text{bid volume (top-5)} + \text{ask volume (top-5)}}$: imbalance restricted to top 5 levels,
- $spread_t = bestAsk_t - bestBid_t$: instantaneous bid–ask spread,
- $directionFeature_t = midPrice_t - lastTransactionPrice_t$: short-term market direction,
- $R_t^k = (r_t, \ldots, r_{t-k+1})$: lagged mid-price returns, with $r_{t-i} = mid_{t-i} - mid_{t-i-1}$ (zero-padded when undefined, default $k = 3$).

This representation ensures that both microstructural variables (spread, imbalance, direction) and execution progress (holdings, time) are incorporated into the decision process, allowing the agent to dynamically balance market impact, timing, and completion risk.

3.2.2 Action space

The action space of the agent is discrete and consists of nine possible actions:

$$\mathcal{A} = \{a_0, a_1, \ldots, a_8\},$$

where each action corresponds to either placing a market order, a limit order of varying size, or holding without trading. Specifically:

$$\begin{aligned}
a_0 &: \text{Market order of size } 10,\\
a_1 &: \text{Market order of size } 20,\\
a_2 &: \text{Market order of size } 50,\\
a_3 &: \text{Market order of size } 100,\\
a_4 &: \text{Limit order of size } 10,\\
a_5 &: \text{Limit order of size } 20,\\
a_6 &: \text{Limit order of size } 50,\\
a_7 &: \text{Limit order of size } 100,\\
a_8 &: \text{Hold (no action)}.
\end{aligned}$$

This design provides flexibility not only in the type of order (market, limit, or hold) but also in the size of the order. As a result, the RL agent can modulate its aggressiveness along two dimensions: (i) choosing between market orders, which ensure execution but increase market impact, and limit orders, which are more patient but risk non-execution; and (ii) adjusting the order size, with larger orders reflecting a more aggressive execution strategy and smaller orders a more conservative one. Thus, the policy learns both when to trade and how aggressively to trade, adapting its decisions to the evolving market conditions throughout the execution window.

3.2.3 Reward Function

The reward function balances cost efficiency, risk control, and execution completion. At each timestep t, the agent incurs a per-step reward based on the *implementation shortfall* (IS) and a rolling variance penalty. Let X_0 be the parent order size, p_0 the entry mid-price, and $\{(p_{t,i}, q_{t,i})\}_{i=1}^{n_t}$ the fills obtained at time t, with average execution price $\bar{p}_t$ and signed side $s \in \{+1, -1\}$. The step shortfall is

$$IS_t = \begin{cases} s\,\dfrac{\bar{p}_t - p_0}{p_0}\,\dfrac{q_t}{X_0}, & q_t > 0,\\ 0, & q_t = 0, \end{cases}$$

where $q_t = \sum_{i=1}^{n_t} q_{t,i}$. The rolling variance is computed over a window of length $w = 5$, which corresponds to five decision steps (roughly 50 seconds in our simulation setup). This choice reflects a balance between responsiveness and stability: a shorter window would make the penalty too sensitive to noise, while a much longer window would dilute recent fluctuations and fail to capture local bursts of market impact. By focusing on short-term variance, the agent is incentivized to avoid erratic execution patterns and to maintain smoother trajectories of implementation shortfall, in line with the emphasis on local risk control in the execution literature (Almgren and Chriss, 2001).

$$\mathrm{Var}_t^{(w)} = \mathrm{Var}(IS_{t-w+1}, \dots, IS_t).$$

The instantaneous reward is therefore

$$r_t = -IS_t - \lambda\,\mathrm{Var}_t^{(w)},$$

where $\lambda > 0$ is a risk–aversion parameter that controls the weight of the variance penalty. In this work, λ is not fixed a priori, but optimized via hyperparameter tuning, as detailed in Section 3.5. If $\tilde{X}_T$ is the cumulative executed quantity, then

$$r_T^{\text{final}} = \begin{cases} c_{\text{not-enough}} \cdot (X_0 - \tilde{X}_T), & \tilde{X}_T < X_0, \\ 0, & \tilde{X}_T = X_0, \end{cases}$$

with $c_{\text{not-enough}} < 0$ set to -100 per share. This structure incentivizes timely completion while minimizing slippage and volatility in execution costs.

3.2.4 Penalty for Incomplete Execution

The terminal cost parameter $c_{\text{not-enough}} = -100$ was chosen to impose a significant penalty on the agent if it fails to complete the execution within the allowed time window. This design encourages the agent to avoid underfilling the order, which would otherwise leave a portion of the trade unexecuted. As discussed by Hafsi and Vittori (2024) and Nevmyvaka et al. (2006), failing to fully execute the order within the episode poses economic risks, including exposure to adverse market movements and increased transaction costs associated with partial fills. This heavy penalty ensures that the reinforcement learning agent develops an optimal execution strategy that prioritizes completing the full order, reflecting realistic trading constraints and operational risks. Without this penalty, the agent might adopt suboptimal policies by prematurely ending the execution, thereby compromising trading performance.

3.2.5 Training setup with GPU

All experiments were executed on a single NVIDIA A40 GPU provided by Tilburg University. Checkpoints were saved every 10 iterations, and the best model was selected according to the mean episode reward. This setup standardizes the evaluation pipeline and ensures that the reported results are not influenced by spurious variability across training runs. The availability of GPU acceleration significantly reduced training time, allowing the agent to complete more iterations within the same computational budget. The training environment was configured using the `markets-execution-v0` environment in ABIDES-Gym with the following parameters:

- Background configuration: `rmsc04`,
- Timestep duration: 10 seconds,
- Execution window: 4 hours,
- Parent order size: 20.000 shares,
- Penalty for underfill: -100 per remaining share.

At the beginning of each episode, the parent order direction (buy or sell) was drawn at random, ensuring that the agent learned to handle both liquidation and acquisition tasks. This prevents the model from overfitting to a single side of the order book and improves robustness in realistic market conditions. At each timestep—corresponding to a 10-second interval in simulated market time—the agent observes the current state of the environment and decides whether to submit a market order, place a limit order (with varying size), or hold without trading. Over the 4-hour execution window, this yields 1.440 sequential decisions per episode. Each action immediately affects the execution trajectory, contributes to the per-step reward, and influences the remaining time and quantity constraints. In this way, the agent continuously balances aggressiveness, patience, and completion risk throughout the episode. The DQN agent was trained for approximately 100,000 timesteps, corresponding to about 70 full execution episodes. Each episode simulates the liquidation or acquisition of a parent order of 20.000 shares over the four-hour horizon, discretized into 1.440 timesteps of 10 seconds each.

3.3 Simulation Environment: ABIDES

3.3.1 Overview

ABIDES (Agent-Based Interactive Discrete Event Simulator), introduced by Byrd et al. (2019), is a widely adopted research platform for simulating electronic financial markets using agent-based methodologies. In contrast to static historical backtesting, ABIDES enables dynamic interaction among heterogeneous agents, causal market impact, and emergent price formation—features that are essential for evaluating reinforcement learning (RL) in trading. The framework supports diverse populations of agents, including market makers, value investors, momentum traders, and noise agents, each with customizable strategies and behaviors (Amrouni et al., 2022; Shi and Cartlidge, 2023; Hafsi and Vittori, 2024).

3.3.2 Event-Driven Architecture

At the core of ABIDES lies its discrete-event simulation architecture. Here, the system state evolves through a sequence of time-stamped, causally ordered events such as order submissions, cancellations, agent wakeups, message passing, and book updates. The kernel manages this global event queue with nanosecond precision, ensuring strict chronological consistency and reproducibility. This event-driven structure makes it possible to capture realistic microstructural dynamics that would be absent in simple replay-based approaches.

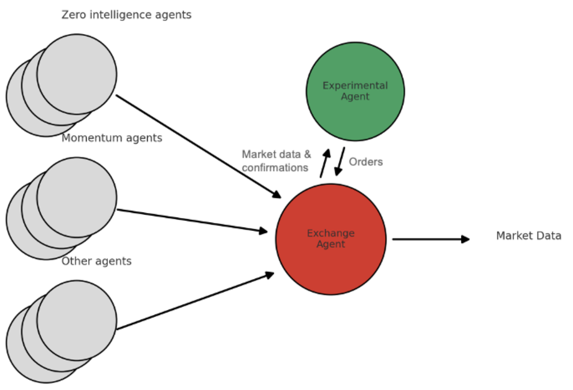

Figure 3.4: ABIDES interaction schema.

Figure 3.4 illustrates the communication flow within ABIDES. All market participants interface with the Exchange Agent, which updates the limit order book, manages trades, and broadcasts market data. The experimental agent, in particular, interacts directly with the exchange, sending orders and receiving observations—a feedback loop that is central to RL benchmarking.

3.3.3 Agent Population and Parameters (RMSC-4)

In this work, we adopt the RMSC-4 configuration, the standard reference setup in ABIDES, which has become a common choice in recent studies (Amrouni et al., 2022; Shi and Cartlidge, 2023; Hafsi and Vittori, 2024). It specifies a realistic electronic market with an exchange agent managing a 10-level limit order book and a 500-stream history, populated by heterogeneous trading agents that generate liquidity, impact, and signals. To better illustrate this structure, Figure 3.5 shows the bid and ask quotes across the first ten levels of the simulated limit order book, highlighting how liquidity is continuously shaped by the interaction of diverse market participants.

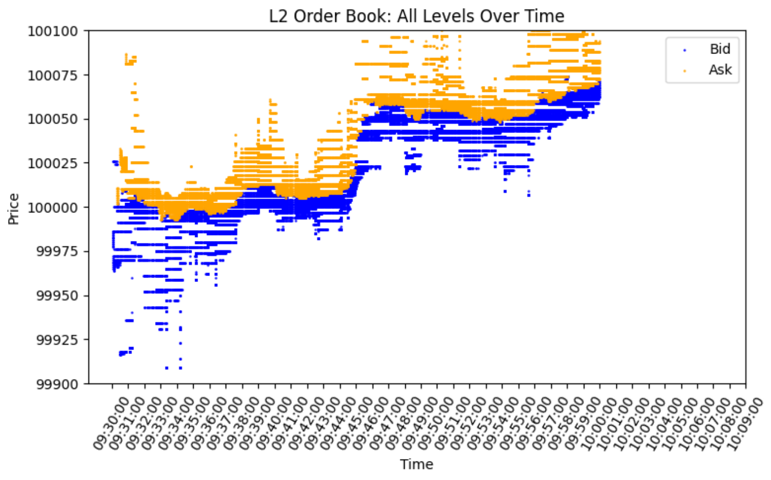

Figure 3.5: L2 order book representation from ABIDES. Bid (blue) and ask (orange) levels evolve over time, reflecting the continuous interaction of simulated agents.

The RMSC-4 population is composed of several agent types:

- Exchange agent: manages the matching engine, maintains the limit order book (10 levels per side), disseminates quotes and trades, and records logs (500 streams). It serves as the central counterpart for all orders and confirmations.

- Noise agents (1000): submit random orders and cancellations, introducing background liquidity and stochastic order flow.

- Value agents (102): trade against a latent fundamental value V_t, modeled as an Ornstein–Uhlenbeck (OU) process with mean $\mu_{\text{va}} = 100{,}000$, mean reversion $\theta_{\text{va}} = 1.67 \times 10^{-15}$, and jump arrivals with intensity $\lambda_{\text{va}} = 5.7 \times 10^{-12}$:

$$dV_t \;=\; \theta_{\text{va}}(\mu_{\text{va}} - V_t)\,dt \;+\; \sigma_{\text{va}}\,dW_t \;+\; J\,dN_t,$$

  where W_t is a Wiener process, N_t a Poisson process for news events, and J the jump amplitude. Value agents tend to buy (sell) when the market price is below (above) V_t.

- Momentum agents (12): react to short-term price trends, amplifying directional moves and clustering aggressive orders in trending phases.

- Market makers (adaptive): quote both sides of the book, manage spreads and depth, and update their quotes dynamically based on inventory, volatility, and order flow. Typical parameters include small order sizes (about 0.025% of volume), narrow tick windows, and wake-up frequencies around one second.

Even with identical parameters, simulations differ depending on the random seed that initializes the pseudorandom number generator. The seed determines the sequence of random events (e.g., noise order arrivals, trade directions, jump timings), so changing it produces distinct price trajectories (see Fig. 3.6). To avoid results that depend on a single "lucky" or "unlucky" path, evaluations should be averaged across multiple seeds to ensure robustness.

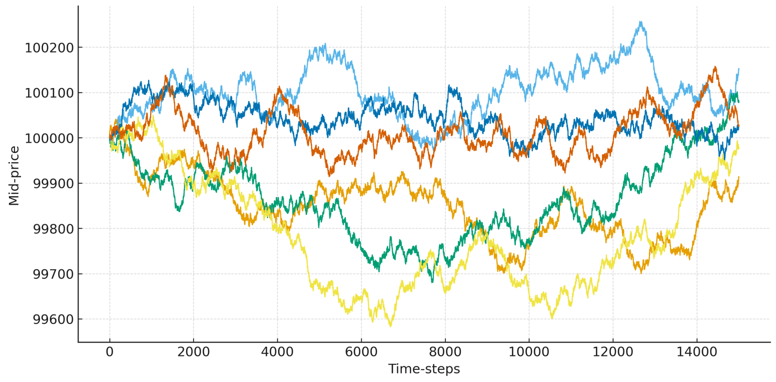

Figure 3.6: Simulated mid-price trajectories generated with different random seeds in ABIDES. Variability across seeds illustrates the stochastic nature of the environment.

3.3.4 ABIDES-Gym Interface

To connect this simulation with reinforcement learning methods, ABIDES is extended through ABIDES-Gym (Amrouni et al., 2022), which exposes a standardized interface to RL libraries such as `gymnasium` and Ray RLlib. At each kernel wakeup, the RL agent receives an observation of the order book and portfolio state, chooses a discrete action (market, limit, hold, sized orders), and is evaluated through a reward function designed for optimal execution. This reward penalizes implementation shortfall, spread costs, and execution variance, thereby aligning the learning signal with established objectives in execution research.

3.3.5 Research Integration

In summary, the RMSC-4 configuration offers a realistic, data-rich, and fully interactive market simulation. Combined with the ABIDES-Gym interface, it provides an ideal testbed for training and benchmarking the Deep Q-Network (DQN) agent studied in this thesis. This environment directly supports the research objective of minimizing transaction costs and optimizing execution performance under realistic microstructural constraints.

3.4 Benchmarks

To evaluate the performance of the DQN agent, we compare it against a set of baseline strategies (*benchmarks*) implemented directly within the ABIDES-Gym environment. These benchmarks cover both simple heuristic policies and one classical execution model widely used in practice.

Heuristic policies

- Aggressive policy: always submits a market order with the maximum size (100). This strategy ensures rapid execution but incurs very high market impact.
- Random policy: selects randomly among the eight available actions (market and limit orders of size 10, 20, 50, 100). This simulates non-strategic behavior and serves as a control baseline.
- Random + No Action policy: extends the random policy by including the hold action (no execution). This introduces further variability and helps test whether the RL agent outperforms a purely stochastic baseline.

Classical execution model

- Time-Weighted Average Price (TWAP): the parent order X_0 is divided into equal slices $n_k = X_0/N$, executed at regular intervals over the horizon T. The realized TWAP price is

  $$P_{\text{TWAP}} = \frac{1}{N}\sum_{k=1}^{N} P_k.$$

  TWAP is the most widely adopted static benchmark and corresponds to the zero-risk solution of the Almgren–Chriss model (Almgren and Chriss, 2001), where $\lambda = 0$.

Together, these benchmarks provide a comprehensive set of references against which the learned policy can be evaluated, highlighting the relative advantages of adaptive execution through reinforcement learning.

3.5 Hyperparameter Tuning for RL Agents: Optuna Integration

3.5.1 Hyperparameter Optimization with Optuna

Training a reinforcement learning agent is highly sensitive to the choice of hyperparameters, which control the learning dynamics, stability, and eventual performance of the algorithm. Parameters such as the learning rate, discount factor, and network architecture can dramatically alter how the agent explores its environment, generalizes patterns, and converges toward effective policies. Manual selection or naïve search methods (grid or random search) are typically inefficient in high-dimensional spaces, and often fail to exploit information from previous trials. To address this challenge, I employed *Optuna*, an open-source hyperparameter optimization framework designed for efficient and automated tuning (Akiba et al., 2019). Optuna relies on three key components:

- *Sampler*: decides which hyperparameter values to try in each trial. Optuna commonly uses the Tree-structured Parzen Estimator (TPE), a Bayesian optimization method that models the distribution of promising and non-promising hyperparameters separately. Instead of searching uniformly, TPE samples new candidates where past trials suggest high reward.

- *Pruner*: terminates unpromising trials early. During training, partial results (intermediate episode rewards) are monitored. If the agent's performance is unlikely to surpass the best observed so far, the trial is "pruned," saving computational resources.

- *Trial*: a single evaluation of the RL agent under a specific hyperparameter configuration. Each trial involves training the agent for a number of episodes and returning an objective score (in my case, the mean episode reward across seeds).

This design makes Optuna particularly efficient:

1. It starts by exploring the space broadly, sampling hyperparameters almost at random.

2. As more trials are completed, the TPE sampler shifts the search toward regions that have historically yielded higher rewards.

3. Simultaneously, the pruning mechanism discards poor configurations after just a few training iterations.

As a result, Optuna adaptively balances exploration (testing new, untried hyperparameters) and exploitation (refining those known to perform well). Compared to exhaustive grid search, it requires far fewer trials to identify near-optimal settings. In this work, Optuna was combined with Ray Tune (Liaw et al., 2018), which provided scalable execution across multiple random seeds and automatic logging. Each trial corresponded to training a DQN agent for 15 iterations, and the optimization was run for 20 trials. The objective function was defined as the average episode reward, since this directly measures execution quality in the trading environment.

Search space. The hyperparameters chosen for tuning reflect the most influential factors in DQN training and in the execution environment:

- Learning rate (`lr`): 10^{-5} to 10^{-2} (log-uniform).
- Discount factor (`gamma`): 0.90 to 0.999 (uniform).
- Train batch size (`train_batch_size`): $\{32, 64, 128\}$.
- Network architecture (`fcnet_hiddens`): $\{[64, 64], [128, 64], [128, 128]\}$.
- Risk aversion parameter (`risk_lambda`): 0.1 to 1.0 (uniform).

Outcome. The tuning identified a configuration that significantly outperformed the default baselines. Table 3.1 summarizes the best parameters found.

Table 3.1: Best hyperparameter configuration obtained via Optuna tuning.

Parameter	Value
Learning rate (`lr`)	0.00237
Discount factor (`gamma`)	0.935
Train batch size (`train_batch_size`)	32
Network architecture (`fcnet_hiddens`)	[64, 64]
Risk aversion parameter (`risk_lambda`)	0.159

The tuned values of `lr` and `gamma` improved stability and accelerated learning, while the optimized `risk_lambda` yielded better balance between transaction cost minimization and risk sensitivity. The pruning mechanism in Optuna proved essential: on average, half of the trials were terminated early, cutting total training time significantly. Overall, Optuna allowed me to identify a robust hyperparameter set with only 20 trials, highlighting the importance of automated optimization in reinforcement learning experiments.

Chapter 4

Experimental Results

4.1 Performance and Convergence

This section presents the empirical results of the DQN agent's training process. We analyze the evolution of the custom reward function, focusing on maximum, mean, and minimum episode rewards. Together, these metrics provide a comprehensive picture of convergence dynamics and robustness under different market conditions.

4.1.1 Reward convergence (maximum)

Figure 4.1 shows the trajectory of the maximum episode reward achieved across training. Initially, the agent reaches only highly negative values due to random exploratory actions. Over time, however, the maximum reward steadily improves and stabilizes near zero. This indicates that under favorable conditions, the agent is capable of approaching an almost optimal execution path, where implementation shortfall and risk penalties are minimized.

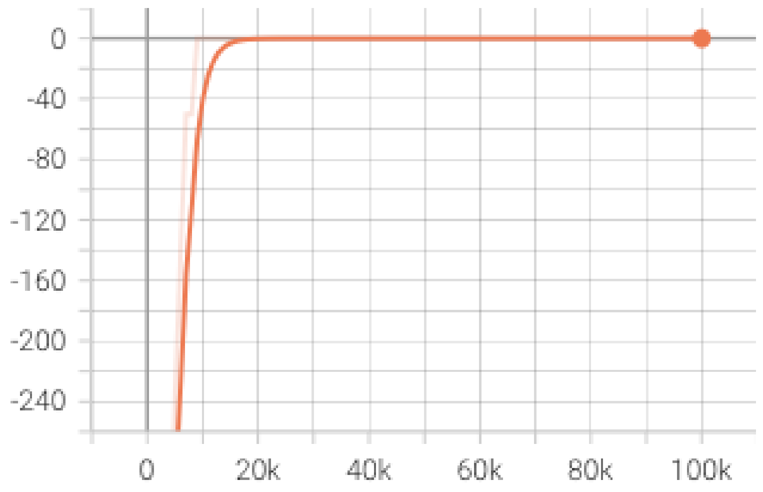

Figure 4.1: Maximum episode reward across training. The curve stabilizes near zero, showing that the agent is able to reach near-optimal outcomes in the best cases.

4.1.2 Reward convergence (mean)

While the maximum reward illustrates the best-case scenario, the mean reward represents the typical performance of the policy. As shown in Figure 4.2, the average episode reward starts from strongly negative values, reflecting the agent's initial inefficiency, and gradually rises to stabilize around -50. This demonstrates that the agent has learned to avoid highly costly actions and can achieve consistent, moderately efficient execution over most episodes.

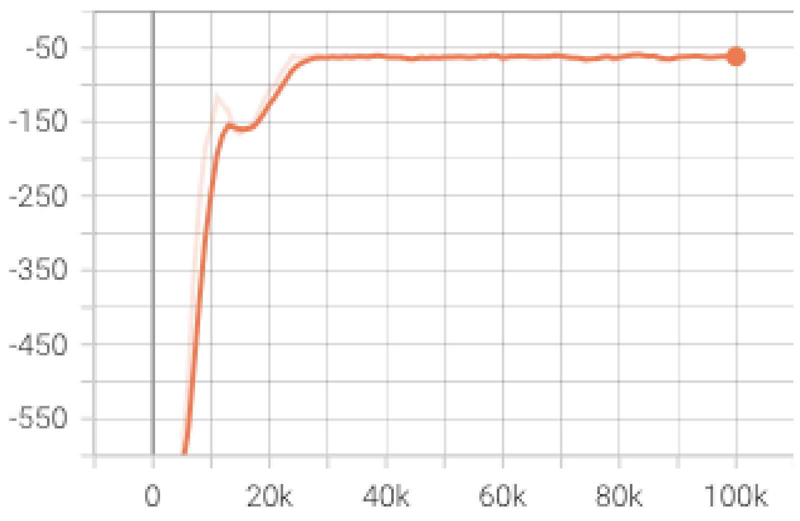

Figure 4.2: Mean episode reward across training. The convergence around −50 reflects consistent improvement across typical scenarios.

4.1.3 Reward convergence (minimum)

The minimum episode reward captures the worst-case outcomes, such as highly unfavorable market conditions or poor exploratory decisions. Figure 4.3 shows that the minimum reward improves over training but stabilizes around −200. This floor demonstrates that, although the agent is robust to extreme scenarios, some downside risk remains inherent to the stochastic nature of the environment.

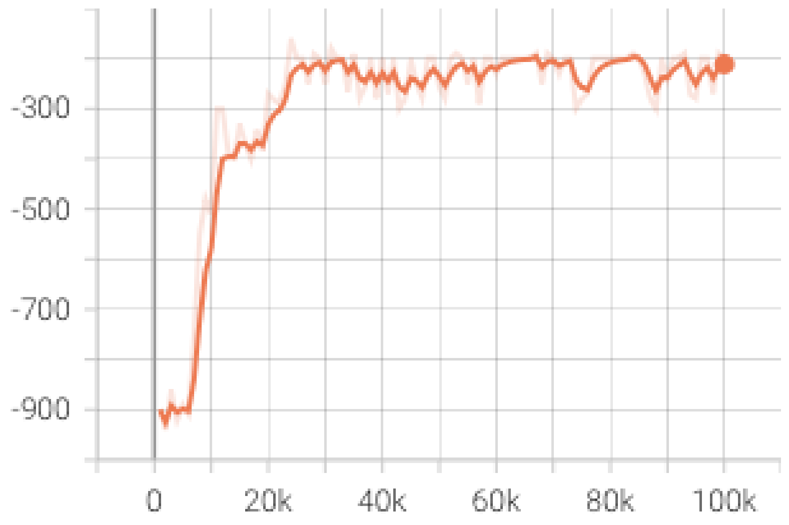

Figure 4.3: Minimum episode reward across training. The curve stabilizes near -200, indicating robustness but also residual exposure to adverse conditions.

4.1.4 Evolution of maximum Q-values

In addition to episode rewards, we analyze the evolution of the maximum Q-value ($\max_a Q(s, a)$) estimated by the agent during training. This metric reflects the agent's expectation of the highest achievable long-term return from any given state-action pair. Figure 4.4 shows that the maximum Q-value initially fluctuates strongly, including a pronounced dip around 20.000 steps, which reflects instability in value estimation during the early exploratory phase. After this period, the curve recovers and stabilizes close to zero, mirroring the stabilization observed in the episode reward metrics.

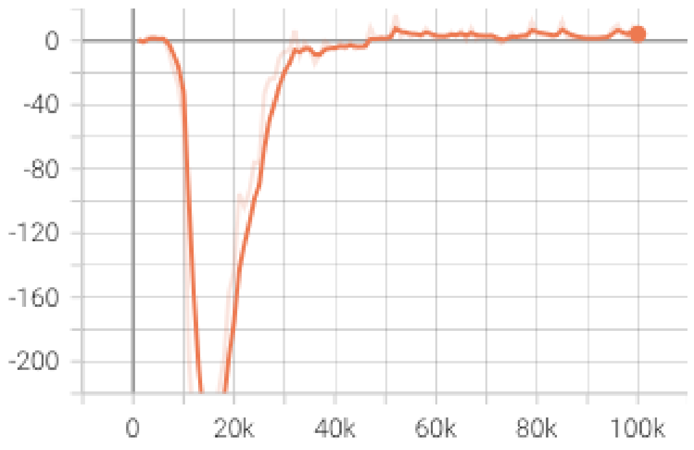

Figure 4.4: Evolution of the maximum Q-value during training. After an initial unstable phase, the estimates converge and stabilize, confirming that the value function has become consistent with the learned policy.

This pattern demonstrates that the DQN agent progressively learns to assign more consistent and realistic value estimates to its actions. The convergence of $\max Q$ alongside the episode reward curves provides further evidence of training stability, confirming that the target network and replay buffer successfully mitigated divergence in value approximation.

4.1.5 Discussion

Taken together, these results demonstrate the convergence of the DQN agent toward a stable and effective execution policy. The maximum reward near zero confirms that the agent can approach optimal execution in the best cases; the mean reward around -50 shows consistent improvement across typical episodes; and the minimum reward around -200 proves that the policy is robust against unfavorable market conditions. The overall pattern—sharp initial improvement followed by stabilization—is characteristic of reinforcement learning in complex environments. After an exploratory phase marked by high penalties, the agent gradually learns to balance execution aggressiveness with cost minimization. The convergence of the three curves demonstrates that the custom reward function successfully guided the agent toward adaptive and risk-aware strategies, outperforming static heuristics such as random execution. Having established the convergence of the training process, we now turn to the out-of-sample evaluation of the agent, analyzing its behavior and comparing it to standard execution benchmarks.

4.2 Behavioral Analysis of the RL Agent

While the previous section focused on the training phase and demonstrated the convergence of the DQN agent, this section shifts to a *backtesting* analysis. Here, the trained policy is evaluated out-of-sample in the ABIDES-Gym environment, using fresh simulations of limit order book dynamics. The goal is to examine how the agent behaves once training is complete, in particular its action preferences and its ability to adapt execution decisions to different market conditions. We first analyze the unconditional distribution of actions across episodes, grouping the nine discrete actions into three interpretable categories (market order, limit order, hold). Then, we study how the agent's execution style changes as a function of the prevailing bid–ask spread, providing insights into its adaptive trading behavior. This backtesting perspective highlights whether the learned policy generalizes well and whether its behavior is consistent with economically meaningful execution logic.

4.2.1 Backtesting setup

In the backtesting phase, the trained DQN agent is evaluated on a standardized execution task. The environment is configured to simulate the liquidation of a parent order of 20.000 shares within a four-hour window, discretized into timesteps of 10 seconds each. This setup mirrors realistic institutional trading constraints and provides a clear benchmark for comparing the learned policy against baseline strategies. Each experiment was repeated over 20 independent episodes, with different random seeds, to ensure robustness of the results.

Table 4.1: Backtesting configuration for the execution task.

Parameter	Value
Parent order size (X_0)	20.000 shares
Execution window	4 hours
Timestep duration	10 seconds
Number of timesteps	1.440
Market background	RMSC-04 (ABIDES reference configuration)
Number of backtesting episodes	20

Figure 4.5 reports the evolution of mid-prices across all backtesting episodes. Each line corresponds to one simulation of the execution task, showing how the underlying market environment evolves during the four-hour window. This visualization highlights the variability of price dynamics the agent faces when liquidating the 20,000-share parent order.

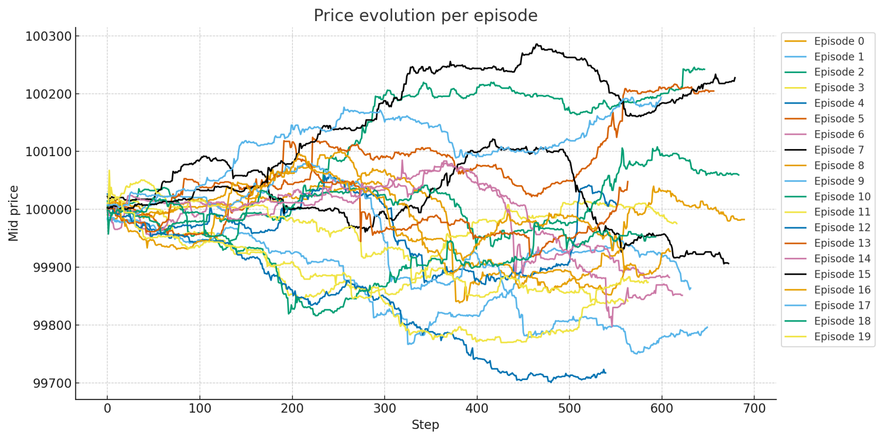

Figure 4.5: Mid-price trajectories across all backtesting episodes. Each line corresponds to one simulated episode of the 20,000-share execution task, highlighting the diversity of market conditions in which the agent operates.

4.2.2 Action preferences of the RL agent

To better understand the trading style learned by the RL agent, we analyze the distribution of actions across all episodes. The original discrete action space consists of nine possible choices, including four market orders of different sizes, four limit orders of different sizes, and a single *hold* action. For interpretability, we group these actions into three categories: *Market Order*, *Limit Order*, and *Hold*. Figure 4.6 reports the relative frequency of each grouped action type.

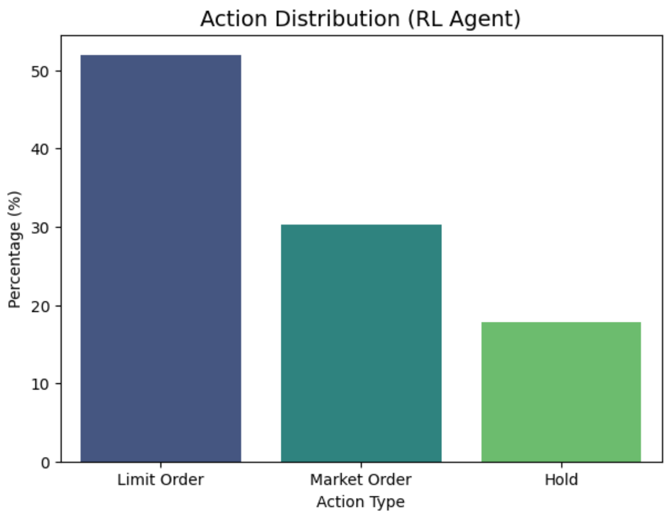

Figure 4.6: Distribution of grouped actions (market orders, limit orders, and hold) executed by the RL agent across all episodes.

The results reveal a clear behavioral preference: the RL agent predominantly relies on *limit orders*, which account for roughly 52% of all actions. This indicates that the agent has learned to prioritize execution strategies that reduce direct transaction costs by avoiding the immediate impact typically associated with aggressive market orders. In contrast, *market orders* represent approximately 30% of the agent's decisions. Their presence highlights that the policy does not solely pursue cost minimization, but strategically mixes aggressiveness when execution certainty becomes more valuable, for instance in the later stages of the execution window or under conditions of low liquidity. Interestingly, the *hold* action is employed in nearly 18% of the cases. This behavior suggests that the agent has internalized the notion of patience: when spreads are wide or when executing at the current moment would incur unfavorable costs, the agent temporarily refrains from trading. This is in sharp contrast to static benchmarks such as TWAP, which systematically execute orders regardless of prevailing microstructural conditions. Overall, the distribution of grouped actions illustrates a balanced policy that blends cost efficiency with risk control. By predominantly using limit orders, the agent minimizes market impact; by occasionally submitting market orders, it ensures that execution progresses toward completion; and by holding when conditions are unfavorable, it avoids unnecessary costs. This adaptive mixture highlights the advantages of reinforcement learning over heuristic strategies, as the agent learns to dynamically trade off between aggressiveness and patience depending on the evolving market state.

4.2.3 Spread-dependent execution behavior

To further investigate how the learned strategy adapts to varying liquidity conditions, we condition the agent's actions on the bid–ask spread. For interpretability, we divide the distribution of observed spreads into three quantile-based categories: *Low*, *Medium*, and *High*. Figure 4.7 reports the distribution of grouped actions when the spread falls in the upper quartile (above the 75th percentile), aggregated across all backtesting episodes.

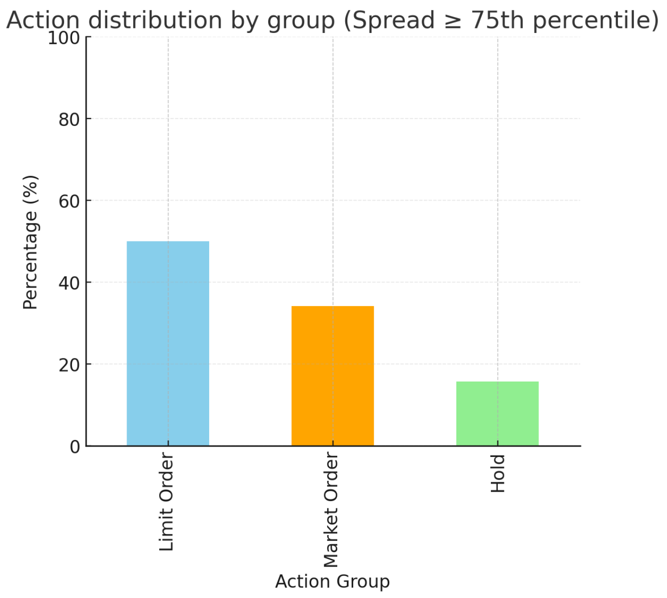

Figure 4.7: Distribution of actions executed by the RL agent when the spread is above the 75th percentile, aggregated across all backtesting episodes.

The figure shows that the agent relies more heavily on *limit orders* when spreads are high, while the use of *market orders* decreases significantly. This behavior suggests that the agent strategically avoids paying the spread cost when liquidity is expensive, preferring patient execution. At the same time, the occasional use of *market orders* guarantees progress toward the completion of the parent order. Finally, the *hold* action remains present, indicating that the agent sometimes postpones execution entirely when market conditions are especially unfavorable.

4.2.4 Case study: Episode 16

While the aggregated statistics provide a global view of the policy, it is also instructive to examine a single episode in detail. Figure 4.8 illustrates the bid and ask price trajectories, together with the actions executed by the RL agent, for a representative backtesting run (Episode 16), chosen at random. The markers denote the type of action taken at each timestep, highlighting how the policy responds to local price and spread dynamics.

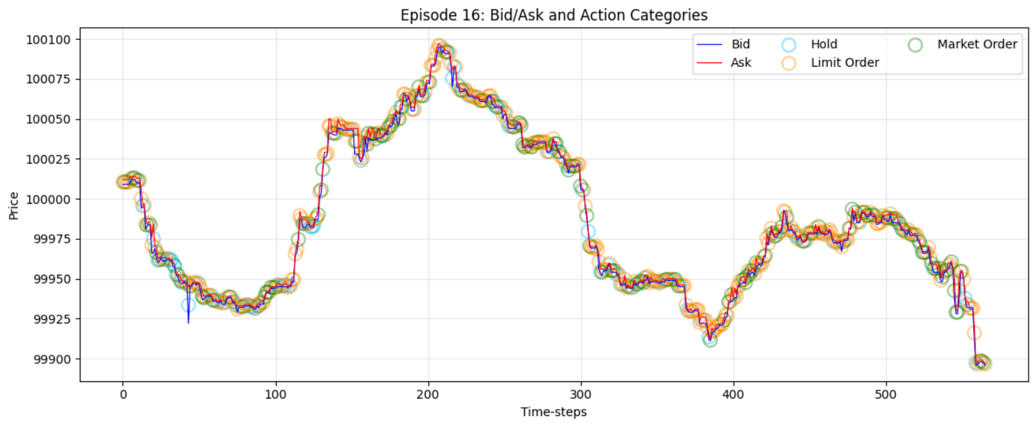

Figure 4.8: Bid and ask price trajectories with the actions executed by the RL agent in Episode 16. The markers denote whether the agent executed a limit order, market order, or hold action at each timestep.

The episode mirrors the aggregate behavior: limit orders dominate, particularly in periods of wider spreads, while market orders are selectively used in moments where execution urgency outweighs cost concerns. The agent also employs the hold action during transient liquidity shocks, demonstrating patience before resuming execution. This micro-level analysis confirms that the strategy learned by the agent is not an artifact of averaging across many runs but a consistent and robust behavioral pattern. To further explore spread-dependent behavior within this episode, Figure 4.9 reports the distribution of grouped actions when the spread exceeds the 75th percentile. The results confirm that, under adverse microstructural conditions, the RL agent strongly favors limit orders, while market orders and holds are used more sparingly.

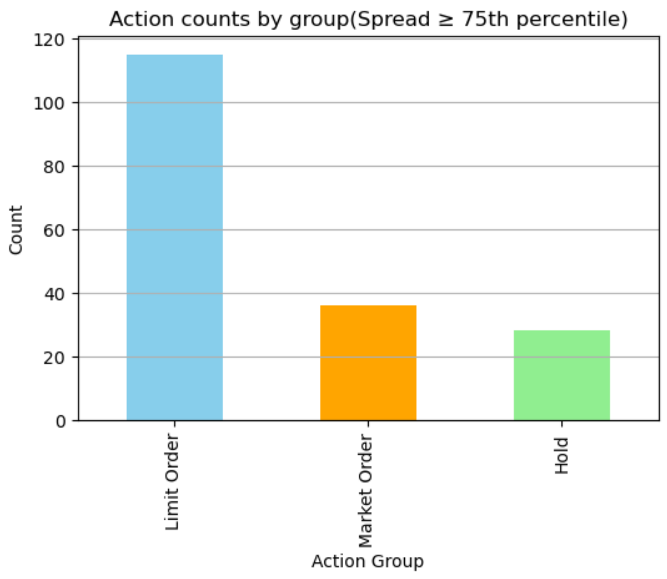

Figure 4.9: Distribution of actions executed by the RL agent in Episode 16 when the spread is above the 75th percentile.

4.3 Performance Comparison with Baseline Strategies

To evaluate the effectiveness of the reinforcement learning (RL) agent, we benchmark its performance against a set of baseline execution strategies implemented within the ABIDES-Gym environment. These baselines include both simple heuristic policies (aggressive, random) and the classical

Time-Weighted Average Price (TWAP) model. By comparing the RL policy with these reference strategies, we obtain a clear assessment of whether the learned agent provides value beyond straightforward heuristics or widely used static execution rules. The evaluation focuses on execution quality across multiple episodes under the standardized backtesting setup introduced in Section 4.2.1. For each strategy, we analyze execution trajectories, implementation shortfall, and overall consistency across market scenarios. This comparative analysis highlights the advantages and limitations of the RL approach, while situating its performance in relation to widely recognized benchmarks. In what follows, we first present the average execution trajectories of all strategies, then examine their distribution of final costs, and finally discuss the observed behavioral differences in trading style.

To assess the effectiveness of the trained DQN agent, we compare its execution performance against the benchmark strategies described in Section 3.4: Aggressive, Random, Random + No Action, and TWAP. This comparison highlights whether reinforcement learning provides a tangible advantage over static or purely stochastic execution policies.

4.3.1 Execution Trajectories and Completion Time

Figure 4.10 shows the average execution trajectories of the RL agent and the baseline strategies, with TWAP.

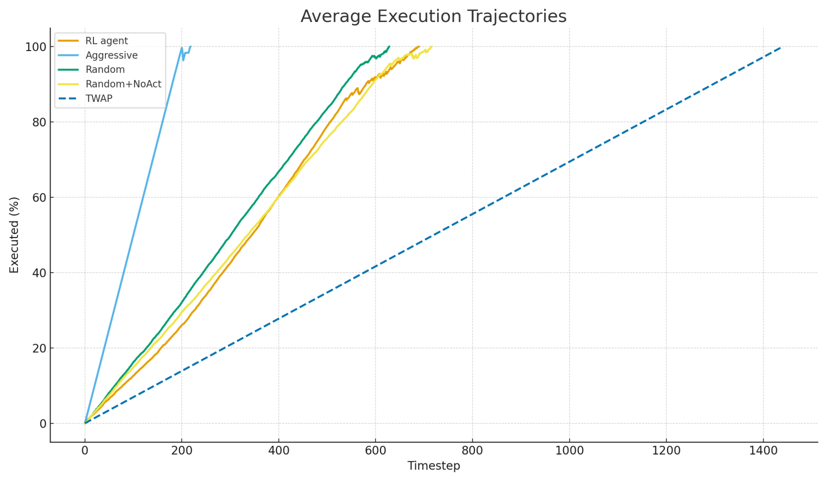

Figure 4.10: Average execution trajectories of the RL agent and baseline strategies. TWAP is shown as a uniform benchmark.

The trajectories highlight distinct execution styles. The Aggressive strategy liquidates the order almost immediately, finishing within the first 200 steps, thereby ensuring immediacy but incurring high market impact costs. Random and Random+No Action exhibit irregular patterns of execution, completing on average around mid-horizon. The RL agent follows a smoother trajectory than the random baselines, requiring less than half of the horizon to complete the parent order, while TWAP executes evenly across the entire four-hour window. Table 4.2 quantifies these dynamics by reporting the average percentage of the 1.440-step horizon required for each strategy to fully execute the 20.000-share parent order. The reported values are averages across the 20 independent backtesting episodes, ensuring that the comparison is not driven by a single simulation run but reflects consistent performance across different market conditions.

Table 4.2: Average completion time relative to the 1.440-step execution window.

Strategy	Completion time (% of horizon)
Aggressive	14%
Random	41%
Random + No Action	46%
RL agent	43%
TWAP	100%

The results confirm that the RL agent achieves a balance between speed and cost: it completes the order in less than half of the available horizon, contrasting with the passive TWAP benchmark and avoiding the overly front-loaded execution of the Aggressive policy. This adaptive pacing demonstrates that the learned policy can dynamically manage execution risk while ensuring timely completion.

4.4 Cumulative Reward Analysis

This section presents the results of the cumulative reward analysis, comparing the RL agent against all baseline strategies under the standardized backtesting setup. The cumulative reward reflects the ability of each policy to minimize execution costs and manage risk throughout the trading horizon, with higher values indicating superior execution quality. Figure 4.11 shows the trajectories of cumulative reward over the entire 1440-step horizon (corresponding to a four-hour execution window with 10-second intervals). The Aggressive strategy exhibits a steep initial increase due to its front-loaded trading style, but then quickly stabilizes, reflecting the high market impact incurred early on. Random and Random + No Action display more irregular paths, with pronounced fluctuations driven by their lack of systematic pacing. The RL agent instead shows a consistently rising curve, reaching higher values than all heuristic baselines. The TWAP benchmark follows a smoother and more gradual trajectory, but its cumulative reward remains below that of the RL agent, reflecting less efficient use of the trading window.

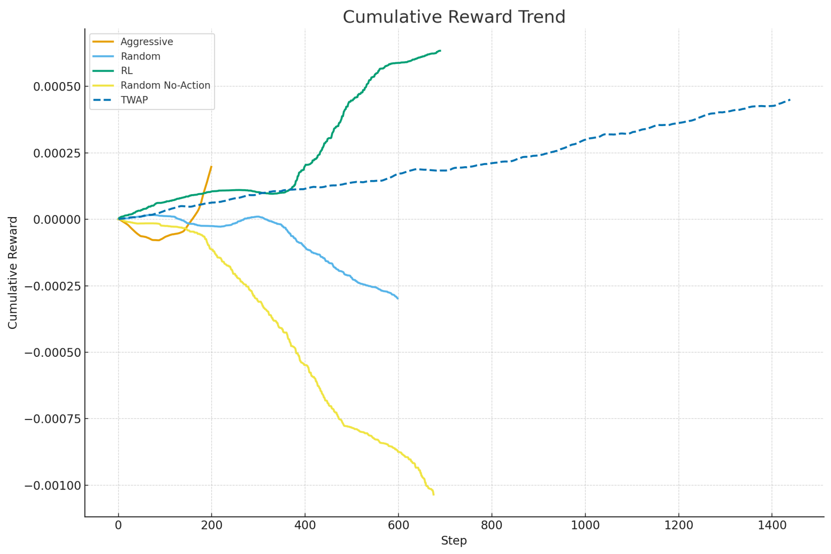

Figure 4.11: Cumulative reward trajectories of the RL agent and baseline strategies over the 1440-step execution horizon. TWAP is included as a benchmark.

It is important to note that the trajectories in Figure 4.11 represent averages across all 20 backtesting episodes, rather than a single run, ensuring that the comparison reflects systematic differences in strategy behavior. To complement these trajectories, Figure 4.12 reports the average cumulative reward achieved by each strategy at the end of the 1440-step horizon. The RL agent clearly outperforms all baselines, including TWAP, achieving the highest overall reward. Aggressive execution secures only modest gains due to the large impact costs of its early orders, while Random and Random + No Action remain heavily penalized by their inefficiency. TWAP delivers predictable but suboptimal results compared to the adaptive strategy of the RL agent.

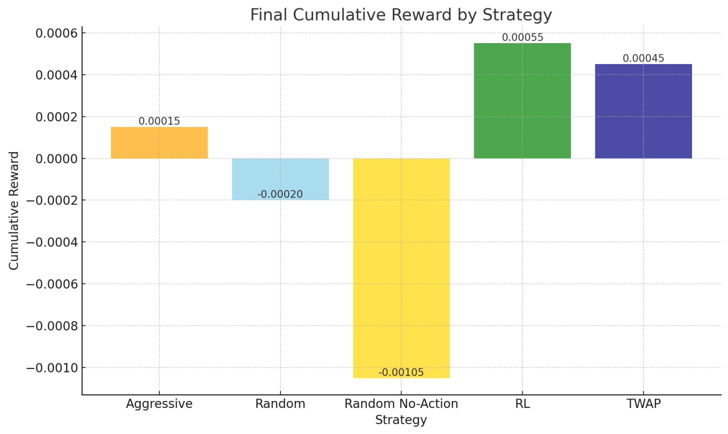

Figure 4.12: Average cumulative reward at the end of the 1440-step horizon across all strategies. Higher values indicate superior execution quality.

Overall, these findings highlight that reinforcement learning not only adapts dynamically to market conditions but also secures superior outcomes in terms of cumulative reward and execution timing, outperforming both heuristic baselines and static benchmarks such as TWAP.

Chapter 5

Conclusion

5.1 Limitations

While the results demonstrate the potential of reinforcement learning for optimal execution, several limitations must be acknowledged. First, the study relies on a single simulated market configuration (RMSC-04) within ABIDES. Although this environment captures many features of a realistic limit order book, it inevitably abstracts away other important dimensions of market microstructure, such as varying tick sizes, cross-asset correlations, or liquidity shocks of larger magnitude. As a consequence, the generalizability of the results to different market settings remains uncertain. Second, the experimental setup fixes key parameters such as the execution window (four hours) and parent order size (20.000 shares). These contraints simplify the analysis but may bias the learned policy toward a narrow class of tasks, limiting its adaptability to heterogeneous real-world trading objectives. Third, reinforcement learning models require substantial computational resources and extensive hyperparameter tuning. Although the Optuna framework helped automate this process, the training remains time-consuming, which may hinder scalability to larger and more complex environments. Finally, the analysis was based entirely on simulated data. While agent-based simulations offer important advantages for controlled experimentation, their outcomes may differ from those observed in live financial markets, where hidden liquidity, information asymmetry, and strategic adversaries play a key role.

5.2 Future Work

Several avenues exist to extend this research and address its current limitations. A natural step is to evaluate the proposed RL framework in more diverse market environments, including different ABIDES configurations and datasets calibrated to specific asset classes such as equities, FX, or fixed income. Testing across a wider range of execution horizons and parent order sizes would also improve robustness and clarify the scalability of the learned policy. From a methodological perspective, future studies could investigate alternative RL algorithms. Policy-gradient approaches such as PPO or SAC may prove advantageous in handling continuous action spaces, while model-based methods could accelerate training and improve sample efficiency. In addition, multi-agent RL frameworks could be explored to capture strategic interactions among multiple liquidity seekers. On the implementation side, reducing the computational burden of training remains an important challenge. More efficient training pipelines, together with transfer learning techniques, could make RL-based execution agents more practical for institutional adoption. Lastly, the ultimate test of robustness will require validation on historical or live market data. Incorporating replay-based environments or hybrid simulation–historical setups would bridge the gap between simulation and reality, ensuring that the strategies remain effective under genuine market conditions.

Bibliography

Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M. (2019). Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery Data Mining*, pages 2623–2631.

Alfi, V., Cristelli, M., Pietronero, L., and Zaccaria, A. (2009). Minimal agent based model for the origin and self-organization of stylized facts in financial markets. *EPL (Europhysics Letters)*, 89(5):58005.

Almgren, R. and Chriss, N. (2001). Optimal execution of portfolio transactions. *Journal of Risk*, 3(2):5–39.

Amrouni, O., Vittorini, E., Hafsi, Y., and Bontempi, G. (2022). Abides-gym: Reinforcement learning environment for limit order book markets. In *Proceedings of the IEEE Symposium Series on Computational Intelligence (SSCI)*.

Amrouni, S., Moulin, A., Vann, J., Vyetrenko, S., Balch, T., and Veloso, M. (2021). Abides-gym: Gym environments for multi-agent discrete event simulation and application to financial markets.

Bertsimas, D. and Lo, A. W. (1998). Optimal control of execution costs. *Journal of Financial Markets*, 1(1):1–50.

Byrd, D., Hybinette, M., and Balch, T. (2019). Abides: Towards high-fidelity market simulation for ai research. *arXiv preprint arXiv:1904.12066*.

Byrd, D., Hybinette, M., and Balch, T. (2020). ABIDES: Towards high-fidelity market simulation for data-driven research. In *Proceedings of the 2020 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation (PADS)*.

Chakraborti, A., Toke, I. M., Patriarca, M., and Abergel, F. (2011). Agent-based model with self-organized traders. *Quantitative Finance*, 11(7):1013–1022.

Coletta, A., Vyetrenko, S., Moulin, A., and Balch, T. (2022). Learning to simulate realistic limit order book markets from data as a world agent. In *Proceedings of the 4th ACM International Conference on AI in Finance*. ACM.

Cont, R., Stoikov, S., and Talreja, R. (2010). A stochastic model for order book dynamics. *Operations Research*, 58(3):549–563.

Freyre-Sanders, A., Guobuzaite, R., and Byrne, G. (2004). Transaction costs and investment performance: Fsa discussion paper. Discussion Paper DP04/3, Financial Services Authority. Accessed August 2025.

Garman, M. B. (1976). Market microstructure. *Journal of Financial Economics*, 3(3):257–275.

Glosten, L. R. and Milgrom, P. R. (1985). Bid, ask and transaction prices in a specialist market with heterogeneously informed traders. *Journal of Financial Economics*, 14(1):71–100.

Gould, M. D., Porter, M. A., Williams, S., McDonald, M., Fenn, D. J., and Howison, S. D. (2013). Limit order books. *Quantitative Finance*, 13(11):1709–1742.

Guéant, O. (2016). *The Financial Mathematics of Market Liquidity: From Optimal Execution to Market Making*. Chapman and Hall/CRC, Boca Raton.

Hafsi, Y. and Vittori, E. (2024). Optimal execution with reinforcement learning. *arXiv preprint arXiv:2411.06389*.

Hamill, L. and Gilbert, N. (2015). Agent-based modelling in economics. *Palgrave Communications*, 1:15016.

Hendricks, D. and Wilcox, D. (2014). A reinforcement learning extension to the almgren-chriss framework for optimal trade execution. *Quantitative Finance*, 14(10):1767–1781.

Huberman, G. and Stanzl, W. (2004). Price manipulation and quasi-arbitrage. *Econometrica*, 72(4):1247–1275.

Karpe, D., B. J. B. T. and Roux, E. (2020). Agent-based simulation for optimal execution using double dqn. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 6905–6912.

Kissell, R. and Malamut, M. (2006). Optimal trading strategies: Quantitative approaches for managing market impact and trading risk.

Kyle, A. S. (1985). Continuous auctions and insider trading. *Econometrica*, 53(6):1315–1335.

Liaw, R., Liang, E., Nishihara, R., Moritz, P., Gonzalez, J. E., and Stoica, I. (2018). Tune: A research platform for distributed model selection and training. In *ICML Workshop on ML Systems*.

Lin, S. and Beling, P. (2020). An end-to-end optimal trade execution framework based on proximal policy optimization. pages 4548–4554.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

Nagy, M. et al. (2023). Reinforcement learning for predictive trade execution. *SSRN Electronic Journal*.

Nevmyvaka, Y., Feng, Y., and Kearns, M. (2006). Reinforcement learning for optimized trade execution. pages 673–680.

Ning, B., H. T. and Zhang, Z. (2018). Double deep q-learning for optimal execution. *arXiv preprint arXiv:1812.06600*.

Obizhaeva, A. and Wang, J. (2013). Optimal trading strategy and supply/demand dynamics. *Journal of Financial Markets*, 16(1):1–32.

O'Hara, M. (1995). *Market Microstructure Theory*. Blackwell, Oxford.

Parlour, C. A. and Seppi, D. J. (2008). Limit order markets: A survey. *The Review of Financial Studies*, 20(4):1467–1492.

Schulman, J., W. F. D. P. R. A. and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Shi, J. and Cartlidge, J. (2023). Efficient deep reinforcement learning for optimal trade execution. *arXiv preprint arXiv:2302.00945*.

Spooner, T., Allshouse, E., Nabi, R., Kamarthi, B., Pinnacle, C., Fonseca, C., Cyranka, J., Pietquin, O., and Prashanth, L. A. (2018). An introduction to deep reinforcement learning for the finance industry. *arXiv preprint arXiv:1809.10147*.

Stigler, G. J. (1964). Public regulation of the securities market. *The Journal of Business*, 37(2):117–142.

Wei, X. et al. (2019). A deep reinforcement learning framework for the financial portfolio management problem. *arXiv preprint arXiv:1901.08740*.

Declaration on the Use of AI Tools

AI-based software was employed exclusively as a writing aid to improve grammar, clarity, and consistency of the text. All research questions, analyses, and written content were developed independently by the author, in full compliance with academic integrity standards.