arXiv:2311.02088v1 [q-fin.CP] 24 Oct 2023

COMBINING DEEP LEARNING ON ORDER BOOKS WITH REINFORCEMENT LEARNING FOR PROFITABLE TRADING

A PREPRINT

Koti S. Jaddu
Department of Computing
Imperial College London
South Kensington, London SW7 2BX
koti.jaddu22@imperial.ac.uk

Paul A. Bilokon
Department of Mathematics
Imperial College London
South Kensington, London SW7 2BX
paul.bilokon@imperial.ac.uk

13 September 2023

ABSTRACT

High-frequency trading is prevalent, where automated decisions must be made quickly to take advantage of price imbalances and patterns in price action that forecast near-future movements. While many algorithms have been explored and tested, analytical methods fail to harness the whole nature of the market environment by focusing on a limited domain. With the evergrowing machine learning field, many large-scale end-to-end studies on raw data have been successfully employed to increase the domain scope for profitable trading but are very difficult to replicate. Combining deep learning on the order books with reinforcement learning is one way of breaking down large-scale end-to-end learning into more manageable and lightweight components for reproducibility, suitable for retail trading.

The following work focuses on forecasting returns across multiple horizons using order flow imbalance and training three temporal-difference learning models for five financial instruments to provide trading signals. The instruments used are two foreign exchange pairs (GBPUSD and EURUSD), two indices (DE40 and FTSE100), and one commodity (XAUUSD). The performances of these 15 agents are evaluated through backtesting simulation, and successful models proceed through to forward testing on a retail trading platform. The results prove potential but require further minimal modifications for consistently profitable trading to fully handle retail trading costs, slippage, and spread fluctuation.

1 Introduction

In 1992, an internet revolution [1, Chapter 1, Page 3] disrupted the financial trading industry when the first online brokerage service provider was launched, E*Trade. This quickly replaced traditional trading over the telephone due to its convenience and faster execution. Naturally, the skill ceiling rose as technology improved. Automated algorithmic trading at high speeds, High-Frequency Trading (HFT), was introduced and became popular in the mid-2000s. This trading method involves a bot constantly identifying tiny price imbalances and entering trades before valuations rapidly corrected themselves, ultimately accumulating small profits over time. In 2020, HFT represented 50% of the trading volume in US equity markets and between 24% and 43% of the trading volume in European equity markets while representing 58% to 76% of orders [2, Page 1]. Although the statistics reveal that HFT is very popular, it hides the fierce competition and immense difficulty- one cannot be perfect in this field. Someone is considered ahead if they find more promising opportunities sooner than their competitors, but these working strategies will not last forever. It is only temporary until a successor arrives, and soon, many will come to surpass. To stay relevant, you must always be the successor. Hence, the emphasis on quantitative research in financial institutions is extensive and in high demand. A portion of such research aims to identify profitable trading strategies

that spot opportunities, enter trades, and manage those trades under a millisecond. Do these strategies exist?

HFT started with hard-coded rules that overlooked the complex nature of financial markets. A justifiable urge to apply Deep Learning to HFT was later birthed, and as hardware improved along with an abundance of data, so did its potential. Lahmiri et al. [3] showcase Deep Learning accurately forecasting Bitcoin's high-frequency price data. Kolm et al. [5] display remarkable results using Deep Learning to correctly predict high-frequency returns (*alpha*) at multiple horizons for 115 stocks traded on Nasdaq. However, only predicting returns is unlikely to help a desk trader because each trading signal's validity would elapse before the trader could input their order. Reinforcement learning can be the key to interpreting this forecasting to execute high-frequency trades because it can learn an optimal strategy when given return predictions at multiple horizons. Independently, Bertermann [4] has trained a few profitable deep reinforcement learning agents using long, mid, and short-term mean-reverting signals as features. The following work combines Bertermann's reinforcement learning with Kolm et al.'s alpha extraction and investigates its potential in a realistic retail trading setting.

Section 1 describes the background and relevant literature to understand the problem while exploring different ideas. The fundamentals of supervised learning will be covered, including their typical pipeline, feed-forward networks, and many network architectures such as Recurrent Neural Networks, Convolutional Neural Networks, and Long Short-Term Memory. Next, reinforcement learning agents such as Q Learning, Deep Q Networks, and Double Deep Q Networks will be touched upon. Limit Order Markets will also be explained to provide more context of the problem. Furthermore, relevant literature will be reviewed which describes different order book feature extraction methods, recent works on using supervised learning and reinforcement learning on the order books, and finally a list of popular performance metrics used to evaluate trading agents. Concepts found in the related work that might not be used in the investigation are also covered in the background for completion and keeping the viewer up to speed.

Section 2 discusses the design and implementation of the solutions, which first describes the data collection pipeline and evaluates the quality of the collected data. Next, the design of the supervised learning and reinforcement learning components (including the three agents) are covered while making certain modifications as recommended in the related work. Finally, the section ends with the testing methodology of the models using backtesting and forward testing.

Section 3 covers the optimisation strategy for tuning the hyperparameters within the supervised learning and reinforcement learning components. All the parameters are explained here, and reasons for setting certain parameters' values without tuning them are justified.

Section 4 evaluates all 15 models after setting their best parameter values using the methodology proposed in Section 4. The performance of the supervised learning and reinforcement learning models are investigated independently through backtesting to support the claims and results observed by their original designers, although modifications were made to try to improve them. This Section also covers the evaluation after combining both models and compares them to a random agent benchmark using statistical testing. The best models were taken through to forward testing and the results are presented. Finally, the agents are explained using heatmaps to investigate what values of input lead to buying and selling behaviours.

Section 5 concludes the findings showing potential and highlights the limitations of this work. Further improvements are proposed to have these algorithms overcome the difficulty of submitting high-frequency trades profitably at the retail level.

2 Background and Related Work

This Section covers the theory required to further understand the problem at hand and break it apart into its components: supervised learning, reinforcement learning, and limit order markets. Relevant published work will also be reviewed which will be built upon.

2.1 Supervised Learning

Machine Learning is a growing field and has been popular in finding statistical patterns in noisy data using reusable blocks in its robust pipeline. This Section introduces the most popular paradigm in Machine Learning and touches on a few architectures of these blocks that will be mentioned in this work. Please refer to [6, 8, 7] for more detail.

There are three Machine Learning paradigms: supervised learning, unsupervised learning, and reinforcement learning. Unsupervised learning is irrelevant to this work, so it will be ignored. Supervised learning is the study of using existing labelled data to automate learning quantitative relationships between its inputs (*features*) to predict the labels of unlabelled data. The terms labels and classes will be used interchangeably, which are one or more values/text stamped to each record of features. Labels can be continuous (*regression*) or discrete/categorical (*classification*).

2.1.1 Pipeline

The pipeline consists of first splitting existing data into a training and held-out testing set (usually of ratio 4:1) to evaluate the model. A model is instantiated, and the data in the training set are fed as batches to update the model- the training process. The training set is separated into its features and labels. The features are passed through the model, and a prediction is made. This prediction is compared with the actual label, and the model is updated through a process called *back-propagation* (please refer to [9]). This encourages the model to make correct predictions when similar features are observed. When the entire data is passed through the model, one *epoch* is completed. Many epochs may be required to train a model. In the end, the held-out test set is used to evaluate the model with unseen data. This is the overall pipeline, but more modifications can improve performance, e.g. splitting the training into a validation set to prevent *overfitting*. This is when the model fits the training data well but generalises poorly and fails to encounter unseen data. Here is a list of *hyper-parameters* that have to be selected before the training process:

- Network Architecture: the number of layers, nodes, and what functions to use
- Loss Function: measures how far the predictions are from the actual values
- Optimiser: the method used to update the parameters of the model
- Learning Rate: the rate at which the model should update its parameters
- Number of Epochs: the number of times iterating the dataset during training
- Batch size: the number of records to expose to the model before each update

2.1.2 Layers in a Network

A network has an input layer, optional hidden layer(s), and an output layer. Data is passed from the input layer to the output layer, and data is manipulated as it propagates through. Layers are only connected to adjacent layers. In a network, one can choose the type of layers and how many are used. The linear layer is the simplest but widely used. It processes the previous layer's output by multiplying that vector with a weight vector, then adding a bias vector, and finally applying a non-linear activation function to allow the modelling of non-linear relationships. If the layer is the network's first (input) layer, it takes in the features as input. These weights and bias vectors are parameters that are learned by the network during training. Popular activation functions include ReLU, Sigmoid, and TanH.

The network designer can also set the number of nodes in each layer, which can be seen as the capacity of the layer. Each node has weighted input and output connections, where the number of connections depends on the number of nodes in adjacent layers. The nodes are usually fully connected but only to nodes in adjacent layers. The computation required to manage an extensive network is demanding, but a compromise is needed to represent more complex functions. Giving a network too much capacity, i.e. too many nodes, can result in overfitting. An illustration of a typical linear network is presented in figure 1.

Convolutional layers in Convolutional Neural Networks (CNN) have reduced parameters compared to linear layers because nodes are not fully connected to nodes in adjacent layers. On top of this, the weights and bias vectors are shared across all nodes in the layer, allowing for the processing of high-dimensional data like images. Ultimately, convolutional layers are good at detecting themes in local regions and are invariant to translation. Pooling operations down-sample image data as it propagates through the network

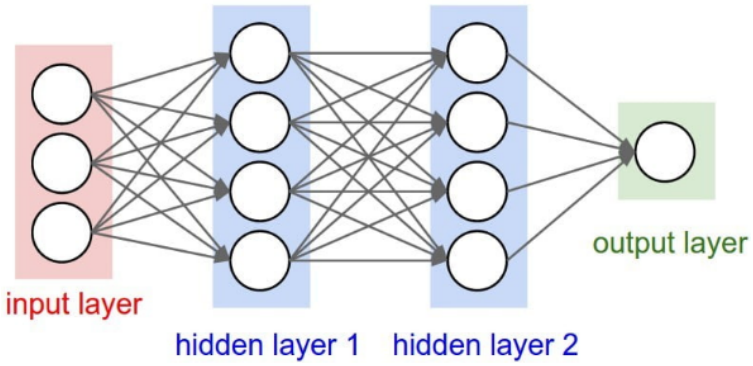


Figure 1: Visual representation of a typical linear network. This network has an input layer with three nodes, two hidden layers each with four nodes, and an output layer with one node. Each line represents the passing of a calculated value to a node in the next layer. This graphic was taken from [10].

to consolidate its learned features and spot global patterns.

Recurrent Neural Networks (RNN) are designed to work with sequential data such as natural language and time series data. This is because they can create a sequence of outputs while propagating the result back into itself to combine it with the next token in the input sequence. After the input is passed, the output will then be the input to the next layer, as mentioned before. A graphic is shown in figure 2.

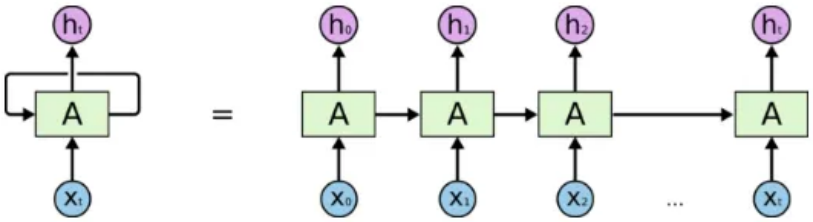


Figure 2: Visual representation of an unrolled Recurrent Neural Network, X is the input sequence and h is the output sequence produced. This graphic was taken from [11].

However, there are many disadvantages to using RNNs. The first is that it has short-term memory, meaning it fails to capture dependencies far apart in input sequences. The second is the gradient vanishing and exploding problem. The longer the input sequence, the more times the weight was multiplied by itself to produce an output. A recurring multiplication with a weight smaller than one eventually becomes too small, and a recurring multiplication with a weight larger than one eventually becomes too big. These extremities make it hard for the network to learn efficiently.

Long Short-Term Memory (LSTM) networks resolve these obstacles by operating three gates. The *forget gate* decides which parts of the long-term memory state to remove. The *input gate* controls what new information to add to the long-term memory state. The *output gate* finally applies a filter combining long and short-term memory with the inputs to return an output. Please refer to figure 3.

2.2 Reinforcement Learning

Reinforcement learning is based on observing rewards from an environment for every action taken and learning behaviours that maximise the reward. The field is particularly popular in Robotics to create agents that operate in the real world. This Section explains the algorithms that will be mentioned in this work. For more information, please refer to [12].

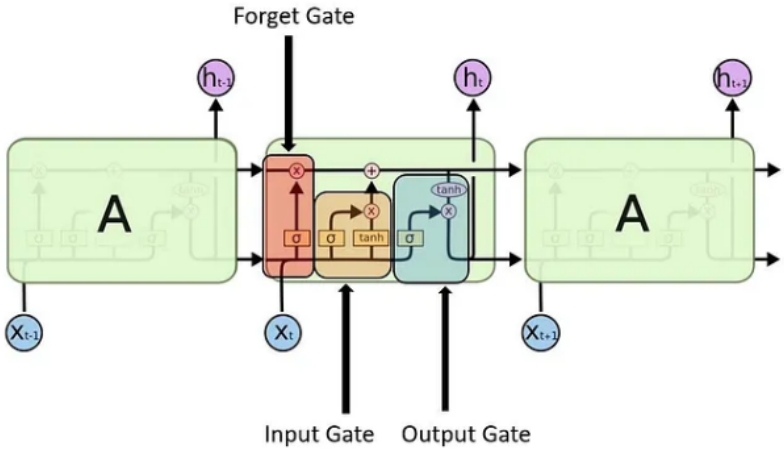


Figure 3: Visual representation of an LSTM cell with its three gates, X is the input sequence and h is the output sequence produced. This graphic was taken from [11].

2.2.1 Terminology

First, here are some key terms that will be used. An action is either discrete (from a finite action set) or continuous (from a real-valued vector), allowing the agent to interact with the environment. A *state* is a minimal but sufficient representation of the environment. It comprises a fixed length tuple with each value indicating the situation of a feature. The environment can be in many different states during an *episode*. An episode is a simulation involving agent interactions with the environment from the initial state to the end state, and the goal is to learn what actions should be taken in what states to maximise the *reward*- desired reactions from the environment observed after every action. Ultimately, learning involves converging to an optimal *value function* or *policy*. A value function $V(s, a)$ takes input a *state-action pair* (the current state and action) and returns the expected return. A policy $\pi(s)$ takes the current state as input and returns the optimal action.

2.2.2 Q Learning

When the dynamics of the environment are not known, *model-free* learning is preferred, which directly estimates the optimal policy or value function. *Monte Carlo* sampling can be used to obtain (state s, action a, reward r, next state s') data which the agent can learn from. *Temporal-difference* (TD) methods combine this sampling with *bootstrapping*- using the estimated values of proceeding states to approximate the value of the current state. Q Learning is a TD method which stores a Q table containing values that estimate the maximum discounted future reward for each action taken at each state. After each reward is observed, these values are updated using the Bellman equation [12, Page 81]. The Q Learning algorithm can be seen in Algorithm 1. Three other parameters exist. The *learning rate* [$\alpha \in (0, 1)$] affects how much each value is changed after each update. The *discounted future reward factor* [$\gamma \in (0, 1)$] controls how much the values in proceeding states affect the current state. The *exploration rate* [$\epsilon \in (0, 1)$] is the probability of performing a random action to explore more state-action pairs. The highest value across the actions in the Q table for a particular state is the (ϵ-*greedy*) action to take.

2.2.3 Deep Q Learning Network (DQN)

Q Learning requires storing a table, meaning that a continuous state space will need to be abstracted to buckets. There is no right way to create these buckets; even after that, a state outside the table's coverage could be visited. Deep Q Learning solves this issue by using a neural network to represent the value function instead of a table, which can generalise well in continuous state spaces. A *target network* helps stabilise the learning process by providing a reference for calculating target Q values and updates less frequently compared to the primary network, which continues to change at every backpropagation step. An *experience replay buffer* D allows samples to be reused and improves computational efficiency by training in mini-batches. Algorithm 2 shows the Deep Q Learning algorithm.

1 Initialise $Q(s, a)$ arbitrarily
2 **repeat**
3 Initialise s
4 **repeat**
5 Choose a from s using policy derived from Q, $\mathbb{P}$(random action) = ϵ
6 Take action a, observe r, s'
7 $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
8 $s \leftarrow s'$
9 **until** s *is terminal*;
10 **until** n *episodes completed*;

Algorithm 1: Q Learning for estimating $\pi \approx \pi_*$ as in [12, Page 153]

1 Initialise replay memory D
2 Initialise primary $Q(s, a; \theta)$ and target $Q'(s, a; \theta')$ networks
3 **repeat**
4 Initialise s
5 **repeat**
6 Choose a from s using policy derived from Q, $\mathbb{P}$(random action) = ϵ
7 Take action a, observe r, s'
8 Store transition (s, a, r, s') in D
9 Sample random mini-batch of transitions (s_i, a_i, r_i, s'_i) from D
10 $y_i \leftarrow r_i + \gamma \max_{a'} Q'(s'_i, a'; \theta')$
11 Perform a gradient descent step on $(y_i - Q(s_i, a_i; \theta))^2$ wrt θ
12 $s \leftarrow s'$
13 Every C steps, set $Q' \leftarrow Q$
14 **until** s *is terminal*;
15 **until** n *episodes completed*;

Algorithm 2: DQN as in [13, Page 6] but with a target network

2.2.4 Double Deep Q Learning Network (DDQN)

A DDQN advances from the DQN by addressing the *overestimation bias*, meaning that the learned Q values are higher than they should be due to the maximum operation used on line 10 in Algorithm 2. This leads to sub-optimal decision-making and is mitigated in the DDQN by using the primary network to evaluate the Q-value of the action selected by the target network. This makes it less likely for both networks to overestimate the Q value of the same action.

1 Initialise replay memory D
2 Initialise primary $Q(s, a; \theta)$ and target $Q'(s, a; \theta')$ networks
3 **repeat**
4 Initialise s
5 **repeat**
6 Choose a from s using policy derived from Q, $\mathbb{P}$(random action) = ϵ
7 Take action a, observe r, s'
8 Store transition (s, a, r, s') in D
9 Sample random mini-batch of transitions (s_i, a_i, r_i, s'_i) from D
10 $y_i \leftarrow r_i + \gamma\, Q(s'_i,\ \underset{a'}{argmax}\, Q'(s'_i, a'; \theta'); \theta)$
11 Perform a gradient descent step on $(y_i - Q(s_i, a_i; \theta))^2$ wrt θ
12 $s \leftarrow s'$ and $\theta' \leftarrow \tau * \theta + (1 - \tau) * \theta'$
13 **until** s *is terminal*;
14 **until** n *episodes completed*;

Algorithm 3: DDQN as explained by Hasselt et al. in 2016 [14]

2.3 Limit Order Markets

This Section describes what the prior theoretical knowledge will be applied to and how it operates, as well as existing methods for transforming raw data for deep feature extraction. The following explanation of Limit Order Markets is based on Hambly's slides [15], which contains an excellent illustrative introduction.

A market is where buyers and sellers meet to exchange goods. A seller will display their asset at a price and wait for a buyer to accept the trade. Contrarily, a buyer can offer a quote and wait for a seller to accept. So how do buyers and sellers make money? You have to make two transactions to start and finish with no inventory (hold no stock). A buyer can make money from buying an asset at price X and then selling it at a higher price Y. The profit is $Y - X > 0$. A seller can also make money, but there is an extra complication. They first would need to borrow an asset, sell it at price X, re-buy it at a lower price Y, and then return the asset to its owner. The profit is $X - Y > 0$. You can therefore make money from buying then selling or selling then buying assets.

Once trading was accessible on the Internet, markets needed to operate in an organised and secure fashion. Limit Order Markets are popular for addressing this. It features a *Limit Order Book (LOB)* which keeps track of a queue of *limit orders* at each discrete price bucket (*ticksize*) and are filled by the counterpart in a first-in-first-out (FIFO) manner. A *buy limit order* is a request to buy **below** the mid-price, and a *sell limit order* is a request to sell **above** the mid-price. The *volume* is attached to each limit order, which is the number of shares the user wishes to buy or sell. Figure 4 shows a visual representation of the ask side (all the sell

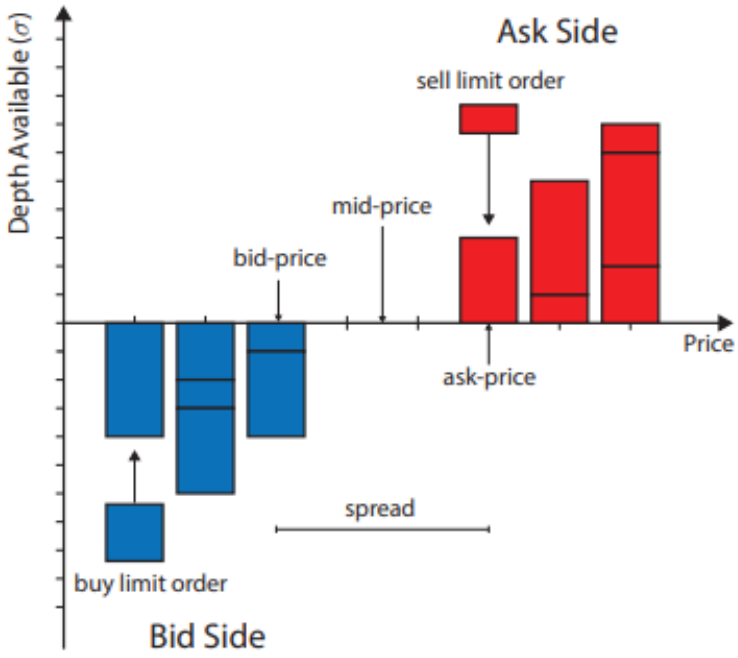


Figure 4: Limit Order Book (LOB) illustration as found in [15, Slide 17]

limit orders- red) and the bid side (all the buy limit orders- blue) along with the corresponding aggregated volume of shares at each price level displayed as *depth available*. The *mid-price* represents the current price of the asset and is calculated in the following way:

$$\mathbf{p}_t^{MID} := \frac{a_t + b_t}{2}, \tag{1}$$

where a_t is the best (lowest) ask price and b_t is the best (highest) bid price at time t. The *bid-ask spread* is the difference between the best ask price and the best bid price $a_t - b_t$. Tighter spreads indicate greater *liquidity* (higher trading activity), which is desirable to ensure limit orders are likely to be filled. When new

limit orders enter the order book, they are added to the end of the queue at the corresponding price level, as shown in figure 4. Once an order is sent, it can be cancelled as long it has not been filled.

Limit orders are filled and leave the queue only at the best bid-ask prices. This happens when an incoming counterpart *market order* of **sufficient** volume is processed; otherwise, the limit order will be partially filled or yet to be filled if it is at the back of the queue. A *market order* is executed immediately, and *market buy orders* are matched with the limit sell orders at the best ask price. In contrast, *market sell orders* are matched with the limit buy orders at the best bid price. Once all limit orders at either the best bid or ask price level are filled, the best bid or ask price moves to the next best bid or ask price level and the mid-price changes accordingly. Similarly, if a new limit order is submitted inside the spread, the mid-price changes because the best bid or ask price has advanced to that new price level. These two scenarios move the market.

2.4 Literature Review

This Section uncovers the related work done in the space of combining deep learning on the order book with reinforcement learning. This includes popular order book feature extraction methods employed as well as reporting the methods and results from using supervised learning and reinforcement learning techniques on the order book. The Section concludes with performance metrics for evaluating trading agents.

2.4.1 Order Book Feature Extraction Methods

Feature extraction is a crucial step when dealing with complex raw data such as the order book. It involves selecting, transforming, and representing the raw data in a more meaningful way, easing the learning process of any model. This is because feature extraction decreases the dimensionality of the data, attempting to mitigate the curse of dimensionality issue [16], which proves an increase in computational complexity when dealing with higher dimensions. However, if the number of dimensions is greatly reduced, then too much useful information is abstracted, so there must be a balance. Feature extraction also reduces the noise in the data as incorporating domain knowledge and expertise allows for keeping what is required for learning and discarding the rest.

As a starting point, here is the raw state of the LOB which will be built upon. It can be abstracted such that each price level has attached a value: the aggregated sum of volumes of limit orders at that price. If we look at the first ten non-empty levels of the order book on each side (bid and ask), the state of the LOB at time t can be written as a vector:

$$\mathbf{s}_t^{LOB} := (a_t^1, v_t^{1,a}, b_t^1, v_t^{1,b}, ..., a_t^{10}, v_t^{10,a}, b_t^{10}, v_t^{10,b})^T \in \mathbb{R}^{40}, \tag{2}$$

where a_t^i, b_t^i are the ask and bid prices at the i-th level at time t and $v_t^{i,a}, v_t^{i,b}$ are the respective aggregated sum of volumes of limit orders at the level. There are many features that can be extracted from the LOB, but three popular methods suitable for predicting mid-price jumps will be explained: price, volume, and order flow.

Price Extraction Methods From initial speculation, removing the volume elements from equation 2 leaves the price components, and so a valid price extraction method could be the following:

$$\mathbf{s}_t^{price} := (a_t^1, b_t^1, ..., a_t^{10}, b_t^{10})^T \in \mathbb{R}^{20}, \tag{3}$$

Although this halves the number of dimensions, there is room for improvement. A simple but useful feature that can be extracted from the LOB is the mid-price, which infers the state of the best bid and ask prices. The calculation for mid-price has been covered in equation 1, and keeping it consistent with the introduced notation- the most abstracted state of the LOB is the following.

$$\mathbf{s}_t^{mid-price} := \frac{a_t^1 + b_t^1}{2} \in \mathbb{R}, \tag{4}$$

This reduces the number of dimensions representing the state of the market from 40 if using ten non-empty bid-ask levels to 1, which is very computationally efficient but removes a lot of information that could be useful. Nevertheless, there exist many trading algorithms that only use the mid-price. Two popular examples include:

- Mean reversion strategies [17] identify deviations from the mean calculated across a period of historical mid-prices and trade hoping price will correct itself towards the mean if deviated above a threshold.
- Momentum strategies [18] identify high volatile movements in mid-price and trade hoping for price to continue in that direction.

Such algorithms require recent historical mid-price data so here is a more appropriate feature extraction method that captures the change in mid-price across ten consecutive timesteps:

$$\mathbf{s}_t^{\Delta mid-price} := \left(\frac{(a_i^1 + b_i^1)}{2} - \frac{(a_{i-1}^1 + b_{i-1}^1)}{2} \,\middle|\, i \in [t-9, t] \right) \in \mathbb{R}^{10}, \tag{5}$$

Volume Extraction Methods Similar to the price extraction methods part, the price components from the raw LOB states shown in equation 2 can be removed, which leaves the aggregated volume values at each non-empty bid and ask level. This would lead to the following extracted feature.

$$\mathbf{s}_t^{volume} := (v_t^{1,a}, v_t^{1,b}, ..., v_t^{10,a}, v_t^{10,b})^T \in \mathbb{R}^{20}, \tag{6}$$

As mentioned, there is room for improvement. If we sum up the quantities in the ask and bid side separately for the first ten non-empty price levels, we get the following.

$$\mathbf{s}_t^{\sum volume} := \left(\sum_{n=1}^{10} v_t^{n,a}, \sum_{n=1}^{10} v_t^{n,b} \right)^T \in \mathbb{R}^2, \tag{7}$$

This shows the number of shares in the observable region on the ask side and the bid side. If there are more shares on the ask side than on the bid side, this indicates that there are more buyers in the market, and so price is more likely to increase. Similarly, if there are more shares on the bid side than on the ask side, this indicates that there are more sellers in the market, and so price is more likely to decrease. To show this more clearly, one can simply subtract the two values and this is the calculation for *volume delta* (VD).

$$\mathbf{s}_t^{VD} := \sum_{n=1}^{10} (v_t^{n,a} - v_t^{n,b}) \in \mathbb{R}, \tag{8}$$

Again, a positive value indicates buying power and a negative value indicates selling power. *Cumulative volume deltas* (CVD) take this idea further and show the difference in bid and ask volumes between consecutive timesteps. This reveals the change in equation 8 over time, which can identify a sudden increase in buying or selling power (very useful during important news releases) and can be calculated as the following.

$$\mathbf{s}_t^{CVD} := \left(\sum_{n=1}^{10} (v_i^{n,a} - v_i^{n,b}) - \sum_{n=1}^{10} (v_{i-1}^{n,a} - v_{i-1}^{n,b}) \,\middle|\, i \in [t-9, t] \right)^T \in \mathbb{R}^{10}, \tag{9}$$

There are many algorithms that use cumulative volume deltas; notable examples consist of volume delta reversal strategies [19], which identify a change of sign in the cumulative volume delta and trade expecting price to reverse direction.

Order Flow Extraction Methods *Order flow* shows the real-time movement of orders entering the LOB. Unlike the mentioned extraction methods, this feature retains both price and volume data, which gives it the potential to provide more insight into supply and demand dynamics. The following explains how to calculate order flow from the LOB as described in [5, Pages 6-7].

Order flow ($\mathbf{OF}_t$) captures the change in the LOB state and, therefore, requires two consecutive tuples, i.e. at time t and $t-1$. It is calculated in the following way:

$$\text{aOF}_{t,i} := \begin{cases} v_t^{i,a}, & \text{if } a_t^i < a_{t-1}^i, \\ v_t^{i,a} - v_{t-1}^{i,a}, & \text{if } a_t^i = a_{t-1}^i, \\ -v_t^{i,a}, & \text{if } a_t^i > a_{t-1}^i, \end{cases} \tag{10}$$

$$\text{bOF}_{t,i} := \begin{cases} v_t^{i,b}, & \text{if } b_t^i > b_{t-1}^i, \\ v_t^{i,b} - v_{t-1}^{i,b}, & \text{if } b_t^i = b_{t-1}^i, \\ -v_t^{i,b}, & \text{if } b_t^i < b_{t-1}^i, \end{cases} \tag{11}$$

where $\text{aOF}_{t,i}$ and $\text{bOF}_{t,i}$ are the ask and bid order flow elements for the i-th level (1 to 10 incl.) at time t. Order flow can be obtained through concatenation:

$$\text{OF}_t := \begin{pmatrix} \text{bOF}_t \\ \text{aOF}_t \end{pmatrix} \in \mathbb{R}^{20}, \tag{12}$$

Order flow imbalance takes this further as it reveals the disparity between the ask and bid order flow components. It can be derived as the following.

$$\text{OFI}_t := \text{bOF}_t - \text{aOF}_t \in \mathbb{R}^{10}, \tag{13}$$

2.4.2 Supervised Learning on Order Books

This Section describes the recent work on the topic of forecasting small price changes (*alpha*) using supervised learning on features extracted from the LOB. Tran et al. [23, 30.5, Pages 1407–1418], Passalis et al. [22, 136, Pages 183-189], and Mäkinen et al. [21, 19.12, Pages 2033-2050] independently but commonly used classifiers to predict price movement, either labelling data into two classes (up or down) or three classes (up, down, or sideways). Labels were predominantly assigned by applying moving averages to price and then using thresholds. Although these methods remove noise from the data, they add modelling parameters that are not desirable in real-world trading applications. For example, the justifications for setting parameters to particular values are probably unreliable because these values are likely to change if there is more data. Kolm et al. [5] address this by changing the problem to regression- a better choice as it removes unwanted assumptions but is more computationally demanding.

Kolm et al.'s [5] design extracts alpha at multiple timesteps (*horizons*)- alpha is the expected change in price, usually after a very short period. Inspired by Cont et al. [20, Pages 47-88] on stationary quantities derived from the LOB (order flow), networks trained on this outperformed networks that were trained from the raw LOB. Kolm et al. [5] took this idea and found that using order flow imbalance as features was a further improvement in aiding the learning process. Regarding the data used for learning, it was standard to source high-quality LOB data from LOBSTER [24].

Figure 5 on the next page presents the results from [5] of the forecasting performance of selective models against the ratio of horizon period over price change. Each model was trained while adopting a 3-week rolling-window *out-of-sample* methodology across 48 weeks using a (1-week validation, 4-weeks training, 1-week out-of-sample testing) structure. The evaluation metric used is out-of-sample R^2 (R^2_{OS}). It is defined in [5, Page 17] as

$$R^2_{\text{OS},h} := 1 - \text{MSE}_{\text{m},h}/\text{MSE}_{\text{bmk},h}, \tag{14}$$

for each *horizon* h, where $\text{MSE}_{\text{m},h}$ and $\text{MSE}_{\text{bmk},h}$ are the mean-squared errors of the model forecasts and benchmark, respectively. The benchmark used is the average out-of-sample return of the stock- 115 symbols were used, and the average across all are presented in figure 5. Overall, the results show that models perform better when using OF and OFI as input rather than raw LOB and that CNN-LSTM extracts alpha most accurately, followed by LSTM-MLP, LSTM, and a stacked LSTM with three layers. MLP stands for Multi-Layer Perceptron, which is a stack of linear layers. The R^2_{OS} (%) is mostly positive, meaning that these networks beat the benchmark.

2.4.3 Reinforcement Learning on Order Books

The related work done using reinforcement learning (RL) on the LOB to execute or simulate trades will be covered. The first large-scale experiments were conducted in 2006 by Nevmyvaka et al. [30], showing the potential of using RL on the LOB. As technology improved alongside increasingly available data and advanced algorithms, RL became more widely used in high-frequency trading. Spooner et al. in 2018 [27] evaluated temporal-difference algorithms (including Q-Learning) in realistic simulations and observed promising performance. To address the trade-off between maximising profits and managing risk in trading, Mani et al. in 2019 [28] incorporated risk-sensitive RL. This agent's decisions are not only led by potential

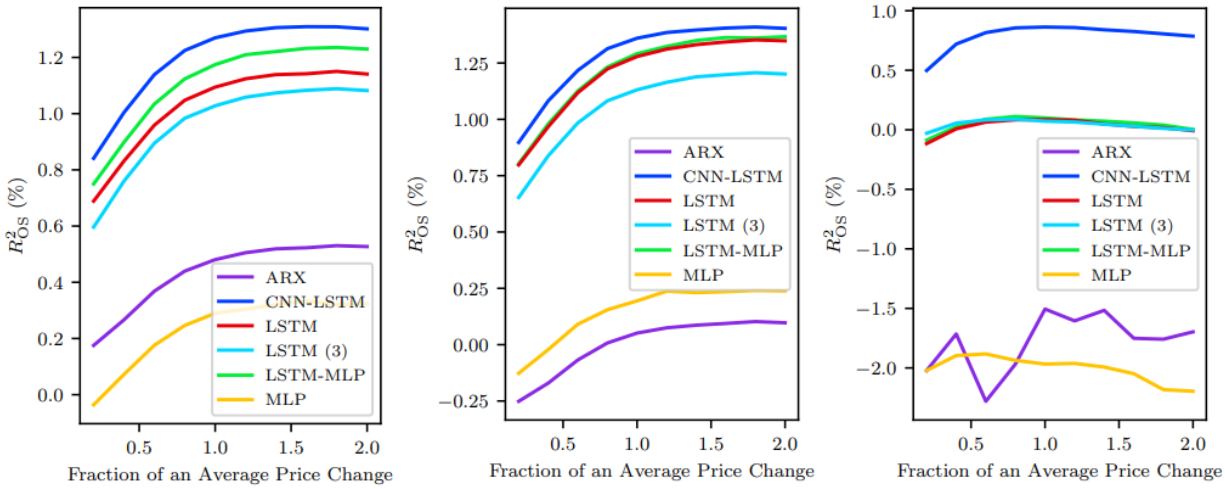


Figure 5: Forecasting performance of different models using OFI (left panel), OF (middle panel), and raw LOB (right panel). Each panel shows the average out-of-sample R^2 against horizons represented as the fraction of an average price change per stock. These graphs can be found in [5, Pages 18 and 35]

profits but are also influenced by potential loss, making it more robust to uncertainty. Despite this, it is considered risky to give an RL agent capital to work with as it lacks explainability, especially in a business context where explainability is essential. Vyetrenko et al. in 2019 [29] address this for risk-sensitive RL strategies by representing the agent's decisions in compact decision trees.

Karpe et al. in 2020 [26] use a Double Deep Q Learning network (DDQN) as well as other RL agents in a realistic LOB market environment simulation. It was astonishing to read that one of their RL agents, in certain scenarios, independently demonstrated the adoption of an existing method: the Time-Weighted Average Price (TWAP) strategy. In brief, the TWAP strategy, according to [25], aims to prevent sudden market volatility (large order impact) by dividing a large order into multiple smaller orders submitted at even time intervals to achieve an average execution price that is close to the actual price of the instrument.

Bertermann in 2021 [4] implemented and compared many RL algorithms, including Q Learning and DQN, for high-frequency trading using the LOB. The features used are long, mid, and short-term mean-reverting signals. Figure 6 displays the results graphically inferred from the original tabular version [4] shown in figure 16 in the Appendices section. From initial inspection, the Q Learning agent has better convergence, and the average profit accumulated after nearly 2,500 episodes of training the Q Learning agent made 35% more than a DQN agent under the same conditions. On top of the standard deviation being 36.3% smaller near 2500 episodes, it implies higher profitable consistency. It is clear that Q Learning is the better algorithm for trading according to these results, which further supports that DQN suffers from instability and overestimation, as mentioned in the Background Section. However, it is important to note that DQN has more changeable parameters, and it could be the case that the parameters were sub-optimal in this study.

2.4.4 Evaluating Trading Agents

There are other ways to evaluate trading agents beyond simply looking at how much money they make during testing (*PnL*). Here is a brief list of common performance metrics accumulated from [32, 31].

- Daily Average Profit/Return: the average of profit or return at the daily level
- Daily Average Volatility: the standard deviation of return at the daily level
- Average Profit/Loss: the ratio of average profit over average loss
- Profitability (%): the percentage of trades that result in a profit
- Maximum Drawdown: the largest peak to trough in the equity curve

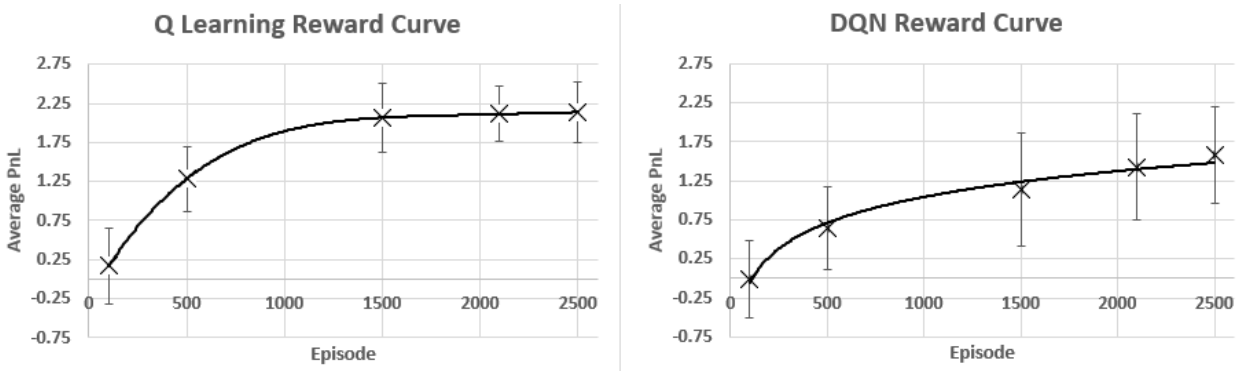


Figure 6: Graphical performance comparison (Average PnL) of Q Learning and DQN inferred from a tabular version provided by Bertermann [4, Page 29]; the error bars are standard deviations

The Sharpe and Sortino ratios will be omitted as they are not insightful for comparing different agents over the **same** test data. They assess the average excess return beyond a benchmark over volatility (downside volatility for Sortino ratio) and, therefore, are proportional to the return while everything else is constant.

3 Design, Implementation, and Testing

This Section covers the design decisions and the implementation of the solution, as well as mentions the process of testing the models. The link to the code repository is *https://github.com/KotiJaddu/Masters-Project*.

The goal is to investigate the combination of deep learning on the order books and reinforcement learning for profitable trading. Two successful studies were selected (one supervised learning and one reinforcement learning) that needed each other in order to be complete, although there was no hesitation in bringing inspiration from other sources. From the different ideas explored in the related literature section, the approach taken forward was adopting Kolm et al.'s [5] very promising alpha extraction to cover the "deep learning on the order books" and sending those outputs into some of the RL agents evaluated by Bertermann [4] to cover the "reinforcement learning for profitable trading". Kolm et al.'s alpha extraction needs Bertermann's reinforcement learning to productise their work, and likewise, Bertermann's reinforcement learning requires quality features compared to lagging mean-reverting signals (despite observing remarkable results).

3.1 Data Collection

This Section discusses the design of the data collection pipeline and evaluates the quality of the collected data. The models to be trained can only be as good as the data fed into them, so a reasonable amount of time was invested into this.

The widely used dataset in relevant literature is sourced from LOBSTER [24]. Although this would also fit well into this work, I wish to integrate the models into my personal retail trading platform (CTrader FXPro [33]) to execute automated trades for realistic simulated testing. This is an award-winning broker with a coding interface that allows access to their LOB. As price action in CTrader is reflected by the orders of its users, the networks will be trained on the LOB data from the platform instead of sourcing it from LOBSTER to maximise accuracy.

CTrader FXPro gives users access to the current state of the LOB but does not provide historical data. Hence, LOB data was collected as early as possible (23/5/2023), saving each LOB state as well as the mid-price per timestep to a CSV file for easy access. Timesteps are defined as changes to the observable LOB states. Storing raw LOB data could bring ethical issues, so the code on CTrader extracts the order flow imbalance features, which is what will be used as input to the networks, and stores that into the CSV file instead as in figure 7. The first ten OFI levels were used by Kolm et al. [5] and should be sufficient to carry out this work.

Time	OFI	Mid Price
22/5/2023 0:23:46:466	[-1, -0.25, -1.875, -1.25, -3.375, -0.25, 0, 0, 0, 0]	1979.35
22/5/2023 0:23:46:560	[-1, -1.75, -1.25, -4.25, -0.25, -1.375, 0, 0, 0, 0]	1979.32
22/5/2023 0:23:46:653	[1, 2, 2.25, 0.25, 2.5, 0.25, 0, 0, 0, 0]	1979.29
22/5/2023 0:23:46:747	[0, 1.75, -2.125, -0.75, -2.25, 1.375, 0, 0, 0, 0]	1979.3
22/5/2023 0:23:46:857	[1, 0.25, 1.75, 2.25, 2.5, 0.5, 0, 0, 0, 0]	1979.3

Figure 7: First five records of the CSV file containing (Time, Order Flow Imbalance, Mid Price) data

As this data extraction should run continuously for ten weeks, this was set up on a Virtual Private Server (VPS) hosted by TradingFX VPS [34], and the files were transferred to local storage every weekend to mitigate the loss upon any technical issue. With regard to the financial instruments that will be scraped, five popular assets were chosen: DE40, FTSE100, EUR/USD, GBP/USD, and XAU/USD (Gold). This covers indices, foreign exchange pairs, and metal.

3.1.1 Analysing Collected Data

Now, analysis will be conducted on the data collected to evaluate its quality. This includes presenting the basic attributes of the data such as mean, standard deviation, and the number of data points. A histogram will also be plotted to look for normality by fitting a bell curve for each dimension in the order flow imbalance vector for each instrument. Normality infers a symmetrical dataset (equal spread of data points on either side of the mean), which improves convergence during learning.

All data was collected on every working day from 23/5/2023 to 3/7/2023 (6 weeks) and from 25/7/2023 to 21/8/2023 (four weeks). There was a technical issue which caused missing data from 4/7/2023 to 24/7/2023, but ten weeks of data was collected in total nevertheless. There is an exception with FTSE100, which is missing one day due to the UK spring holiday (29th of May).

Table 1: Basic Attributes of the Collected OFI Data

Attribute	XAUUSD	GBPUSD	EURUSD	FTSE100	DE40
Total Data Points	13,356,795	6,177,143	4,513,533	1,647,380	1,535,992
Daily Avg. Count	267,136	123,543	90,271	33,620	30,720
Mean OFI value	0.1218	0.0846	0.0775	0.1052	0.0203
Standard dev.	0.4197	0.4060	0.4136	0.4848	0.3727
% Positive	48.0	47.3	48.0	37.6	41.3
% Negative	20.4	26.0	28.3	23.5	32.1
% Zero	31.6	26.7	23.7	38.9	26.6

Table 1 shows the basic attributes of the collected data. The number of data points will be sufficient to train the small-scale models for each instrument. It is interesting to see that all the means are positive, indicating that the ten weeks of data recorded could be bullish-biased. This is supported by all the OFI containing almost twice as many positive values as negative values, although this might not necessarily directly correlate to positive price movement. However, the standard deviations being almost five times more than the means infers that there is sufficient coverage of negative values in the data.

To better understand the spread of the data, figures 17-21 in the Appendices Section show the histograms of OFI values (normalised and then scaled) at each of the ten levels for each instrument as well as a fitted normal distribution on top. Overall, the spread is favoured towards the right side of the mean, and the fitted normal distribution is somewhat appropriate as one could observe a peak near the mean and then tails towards the extremities. These graphs show sufficient coverage of positive and negative values, enough to train the network for alpha extraction. The OFI distributions for DE40 (figure 20) are the best as almost all levels show a valid fit with the exception of levels 1 (bimodal), 5 (skewed), and 6 (bimodal).

Tables 17-21 in the Appendices Section go into more detail showing the mean, standard deviation, and percentage of positive, negative, and zero values for each OFI level in the collected data for every instrument. As expected, the proportion of zero values increases the higher the OFI level because more updates to the LOB happen at lower levels near the mid-price, especially at levels 2 to 6.

Both supervised learning and reinforcement learning components will use 80% of the data for training, 10% of the data for validation (for hyperparameter tuning), and the final 10% of the data for testing.

3.2 Supervised Learning

This Section discusses the decisions made during the integration of Kolm et al's [5] alpha extraction into this work. Overall, order flow imbalance features will be collected, which will be used as input to a supervised learning regression model that will predict the change in mid-price (alpha) for the next six horizons. The reason for choosing six is that Kolm et al. [5] observed that price prediction accuracy falls off after six horizons. The output of the code is to save the model to disk for use in the reinforcement learning part. At the end of this Section, the quality of alphas to be set as labels to OFI features are evaluated to ensure good coverage.

The learning of a model for each instrument will be written as a reusable function using Python- the Python programming language was chosen because of the vast number of machine learning libraries and available documentation. Furthermore, the PyTorch library [35] will be used, which has more functionality than needed for this work. With regards to the code, it will be well commented to ensure readability such that any developer can make further modifications with ease. It will also make use of the GPU with CUDA [36] during training to save time.

3.2.1 Pipeline

Apply preprocessing to data
Split data ratio (7: 1: 2) into training T, validation V, and testing H sets
Initialise hyperparameters
Initialise model M and optimiser *Opt(Method, Learning Rate)*
$k \leftarrow 0$ # for early stopping
repeat
  repeat
    $x, y \leftarrow$ *next batch from* T # features x and labels y
    $y' \leftarrow M(x)$ # *predictions*
    Loss $\leftarrow L(y', y)$
    Opt.backpropagate(Loss) # updates weights and bias values in M
  until T *fully iterated*;
  if *error of* M *on* B *has improved from previous best* **then**
    $k \leftarrow 0$
  end
  else
    $k \leftarrow k + 1$
  end
until $k =$ *maximum patience tolerable or maximum epochs reached*;
Evaluate M with H and process results

Algorithm 4: Supervised Learning Algorithm

Algorithm 4 will be the pipeline as explained earlier with a further enhancement from Kolm et al. [5]. It is normally a difficult task to figure out how many epochs are needed to train the model for optimal results. If the number of epochs is too small, then the model will be underfitting, and if the number of epochs is too large, then the model will be overfitting- both of which are undesirable, and a lot of time-consuming trialling is required. Early stopping will terminate the training when it thinks the model is starting to overfit. It does this with a validation set, as in algorithm 4, by checking how many epochs in a row the model has not improved since its previous best score, and if this number of epochs exceeds the maximum patience tolerable, then the learning will terminate. However, a problem arises when the maximum number of epochs is not limited, which is very long training times. A solution is to set a limit to the maximum number

of epochs tolerable.

As recommended by Kolm et al. [5], L2 regularisation was adopted to prevent overfitting. This adds a small penalty term to the loss function and encourages the weights to be small. From initial experimentation, using LSTM and LSTM-CNN layers was not beneficial compared to using just a multi-layer perceptron (MLP). The reason for this could be that these complex networks are difficult to train with limited hardware. Although not as successful as LSTM and LSTM-CNN networks, Kolm et al. [5] show that a MLP also produces promising results, so this will be the backbone architecture of the regression networks.

There are many hyperparameters in algorithm 4 that require tuning. These are batch size, optimiser method, learning rate, network architecture, loss function, maximum epoch number, and early stopping patience parameter. Although optimisation is an important topic for maximising model accuracy, the methodology for tuning these variables will be covered in the next Section - Optimisation.

Regarding the preprocessing on line 1, limiting the data to remove periods of low trading participation (pre and post-market) came to mind. However, the OFI should also indicate low volatility, which the network should pick up. Keeping all trading periods in the data is also a suggestion in the further works of Karpe et al. [26] for their work, so nothing was removed.

3.2.2 Analysing Alphas from Collected Data

The preprocessing also involves calculating alphas from the data and setting them as labels to the OFI features. Please see figure 22 in the Appendices Section for tabular outputs. The attributes of the alphas will be analysed to evaluate the spread of the labels in the data. The process will be similar to the previous evaluation of data and will be applied to each instrument.

Table 2: Basic Attributes of the Calculated Alphas

Attribute	XAUUSD	GBPUSD	EURUSD	FTSE100	DE40
Mean value (Price)	-2.85e-05	1.15e-08	-4.25e-09	-1.42e-03	-1.79e-03
Standard dev. (Price)	4.80e-02	3.16e-05	2.74e-05	5.30e-01	1.40e+00
Tick Size	0.01	0.0001	0.0001	0.1	0.1
Mean value (Pips)	-2.85e-03	1.15e-04	-4.25e-05	-1.42e-02	-1.79e-02
Standard dev. (Pips)	4.80e+00	3.16e-01	2.74e-01	5.30e+00	1.40e+01
% Positive	37.3	38.7	37.2	39.5	35.2
% Negative	37.9	38.6	37.4	39.5	35.2
% Zero	24.8	22.7	25.4	21.0	29.6

Table 2 presents the attributes of the calculated alphas. The tick size is the smallest movement that the best bid or ask level can move in price units, which is equal to 1 pip. Converting price movements to pips will allow to standardise the values for comparison across all instruments. From this data, the instruments can be ordered (descending) according to standard deviation, which is the volatility and risk: DE40, FTSE100, XAUUSD, GBPUSD, and EURUSD.

Furthermore, the table shows that the mean alpha movement in terms of pips is close to 0, and on top of a much larger standard deviation, there seems to be good coverage of positive and negative values. This is further supported by the proportion of positive and negative values being almost identical for every instrument, which is ideal because the network can learn bullish and bearish behaviour equally. It is also important for the network to learn when price will not move, so a good proportion of zero movement is also very useful. Tables 22-26 in the Appendices Section go into more detail for each horizon level in the collected data for every instrument. An interesting pattern followed by all instruments except XAUUSD is that price usually oscillates back and forth between adjacent price levels. This can be seen through the percentage of zero alphas peaking at every other horizon in table 26.

3.3 Reinforcement Learning

This Section discusses the decisions made during the integration of Bertermann's [4] RL agents (Q Learning and Deep Q Learning) into this work. A Double Deep Q Learning Network (DDQN) will also be investigated

as it showed successful results according to Karpe et al. [26] and addresses the problems with the DQN as discussed in Section (1.2.4).

The code will be written in the Python programming language using PyTorch wherever applicable, as mentioned in the previous Section. There will be three scripts dedicated to representing each agent: Q Learning, DQN, and DDQN. This separates the agents such that any can be used for future work, which is difficult with the alternative approach consisting of merging all agents into one condensed file and passing the agent to use through the command line. However, this alternative approach would be more efficient as each file includes the learning process and testing method on unseen data to evaluate the performance. Algorithms 1, 2, and 3 will be used to implement the RL agents. There are many hyperparameters that will need to be tuned, which will be discussed in the next Section.

There are two methods for using market data to train these RL agents: using offline data (see *backtesting* Section 2.4.1) and using online data (see *forward testing*). Offline data involves iterating through historical data for training, which is faster to process compared to using online data that uses real-time data given from an environment. As the supervised learning component requires the storing of historical data, offline data is therefore available to simulate the markets for learning. The implementation involves coding the backtesting process, using algorithm 5, in a separate Python script, which will be used by all RL agents. Taking inspiration from OpenAI's Gym Environment [37], the backtesting is formatted as an environment (in the utils.py script) that sends the agent the next state and reward when given the current state and action while applying retail-level commissions to the profit or loss for each trade. Forward testing is important for realistic simulation, so CTrader's built-in forward-tester (algorithm 6) will be used. This involves creating a web application using Flask [38], which will use the models to send signals to the platform upon receiving POST requests. Another way was to edit and read text files, but this would slow the process as both applications cannot access the same file at once.

There are two ways to use the historical data for training. The first is using the predicted alphas from the supervised learning component (alpha extraction) as input to the RL agent. The second is calculating the actual alphas stored in the historical data and using that as input. The first method allows the RL agent to train to the inaccuracies of the alpha extraction model, whereas the second trains independently to the alpha extraction- which relies on the alpha extraction to be accurate for the RL agent to fully utilise its training. The second method was selected to avoid training bias. Due to limited data, there will be an overlap of data used to train both the alpha extraction and RL agents. This means that the alpha extraction will be more accurate when passing through data it was trained on, and its accuracy will likely differ on unseen data- this will confuse the RL agents. Furthermore, keeping the training process independent allows for the alpha extraction to be changed or improved without the RL agents being affected.

The agent will always hold an active position in the market at all times whether, it be a buy or a sell position. This is to keep the agent simple and show potential for this approach. The agent can only alternate the direction upon receiving a reversal signal; therefore, it should have enough knowledge to be able to weigh the potential reward for reversing the current position, given the transaction costs in doing so. Hence, the state space should also contain the state of the current position on top of the alphas as input so it can figure out whether it is worth reversing the current position. There will be seven states in total (six alphas and one current position). This is an extension to Bertermann's [4] setup.

From initial experimentation, using exploration decay was found to be important, which decreases the amount of exploration done by the agent over time while emphasising high exploration during early stages of training. This was recommended by Bertermann [4]. On the other hand, a decay in learning rate did not show any improvements in learning and, in most cases, worsened performance. L2 regularisation was also incorporated in the DQN and DDQN models to prevent overfitting by encouraging the weights to be small.

3.4 Testing

This Section covers the methodology of testing each model to evaluate its performance fairly for comparison. Backtesting and forward testing will be explained.

3.4.1 Backtesting

Trading strategies or automated trading bots need to be evaluated in a risk-free but realistic environment before it is validated for live trading with real money. *Backtesting* is the process of simulating the market with historical data to track the results of buy/sell signals according to a strategy. The actual outcomes had these signals been taken as trades will be used to assess the agent's performance at the end of the simulation.

Algorithm 5 shows a backtesting routine for our agent that alternates between continuously holding an active buy or sell position on an instrument. All the trade signals, as well as their results, are logged in T, which will be used to calculate the performance metrics needed to evaluate the agent. As CTrader does not provide historical LOB data, backtesting will need to be implemented independently using code. In line 14, transaction costs include commissions and spread (the difference between the best ask price and best bid price).

```
1  Load agent A
2  Load backtesting data D as a list of tuples with form (OFI, Mid-price)
3  Initialise empty trade log T
4  Initialise previous action a' ← NULL
5  Initialise current trade t ← NULL
6  Initialise current state s ← next(D)
7  repeat
8  |  a ← A(s)  # buy or sell signal
9  |  if a = a' then
10 |  |  Update t using s
11 |  end
12 |  else
13 |  |  Append non-NULL t to T
14 |  |  Reinitialise t with (s, a) and transaction costs
15 |  end
16 |  s ← next(D)
17 |  a' ← a
18 until D is fully iterated;
19 Use T to calculate performance metrics
```

Algorithm 5: Single Position HFT Backtesting Algorithm

There are many events that are difficult to replicate in this environment due to shifts in market volatility from news releases and events. Some are the following:

- Fluctuating spreads from traders being either passive or active during news
- Slippage is when the execution price is not the expected price of entry
- Being partially filled or not being filled if the entry price is hit but reverses

3.4.2 Forward Testing

When a trading strategy has been proven to be promising, usually from good performance during backtesting, one will consider passing their agent through *forward testing*. This is a form of testing that connects an agent to a platform, like CTrader, where it can process live unseen data and submit real orders. It can trade with either real money or fake money (known as *paper trading*).

This is more realistic as the platform will notify the agent when an order has been successfully filled (with the exception of paper trading as there is no real order), filled at a different price compared to the expected price of entry (slippage), and affected from fluctuating spreads. The results from forward testing will, therefore, be more accurate compared to backtesting. From inspection, if forward testing is more accurate compared to backtesting, then why do backtesting at all? Why not only do forward testing? It is because forward testing is more time-consuming as it is using live data compared to iterating through historical data like in backtesting.

Similar to before, Algorithm 6 shows a forward testing method for our agent that alternates between continuously holding an active buy or sell position.

```
1  Load agent A
2  Acquire live data feed s as a tuple (OFI, Mid-price)
3  Initialise empty trade log T
4  Initialise previous action a' ← NULL
5  repeat
6  |  a ← A(s)  # buy or sell signal
7  |  if a is not a' then
8  |  |  Append result of t if it exists to T
9  |  |  Close position t if it exists and open position t aligned with a
10 |  end
11 |  Wait for an update from the live data feed and store the new state in s
12 |  a' ← a
13 until terminated;
14 Use T to calculate performance metrics
```

Algorithm 6: Single Position HFT Forward Testing Algorithm

The main changes consist of using the live data feed instead of historical data (line 2), closing and opening a new trade on the trading platform (line 9), and waiting for a new incoming update from the data feed (line 11).

4 Optimisation

This Section discusses any experimentation and optimisation undertaken to improve the solutions for the supervised learning and reinforcement learning components. This will primarily be via hyperparameter tuning to improve the models after starting with settings proposed by Kolm et al. [5] and Bertermann [4] as a baseline for certain parameters. Hyperparameters already discussed in the Design Section will reappear for completion. The outcome of the tuning will be presented in the next Section.

4.1 Supervised Learning Tuning

Here is a list of all the hyperparameters that will need to be tuned for the alpha extraction component.

- Network architecture → includes the type of layers that should be used in the network, the number of hidden layers, the number of nodes in each hidden layer, and the activation function to apply after each layer. These control how data is processed, which needs to be appropriate to the task. They also configure the capacity of the network- too large capacity leads to overfitting, and too small capacity leads to underfitting.
- Optimiser method → defines the way to optimise the model's parameters during training to reduce the error and converge towards the optimal configuration. Choosing the right optimiser is important as it impacts the convergence speed and the final performance of the model.
- Learning rate → is the step size determining how much the model's parameters are adjusted- too small rates result in slow convergence, whereas too large rates will fail to converge due to overshooting the optimal solution.
- Batch size → is the number of records in the training data used to update the model's parameters in one go. This adds noise to the model, which helps it generalise to unseen data. Small batch sizes inject too much noise into the model, which slows training, while large batch sizes reduce the regularisation effect and are computationally expensive.
- Loss function → is used to compare the predicted value to the actual value and compute an error. An inappropriate function will result in the model learning and prioritising incorrectly.
- Early Stopping Patience → is used to stop the training when the model begins to overfit. If this value is too low, then the model can underfit, but if it is too large, then the model can overfit the data.
- Maximum number of Epochs → the number of epochs before automatically terminating the training process. This is to avoid very long training times when only relying on early stopping.

4.1.1 Justifying Values for Certain Parameters

There are certain parameters where their optimal values can be justified using existing knowledge. Hence, they will not need to be tuned unless they are a starting point from existing literature. Here is a list of hyperparameters where their values are justified among popular alternatives.

- Type of layers (Linear, LSTM, LSTM-CNN) → Initial experiments showed that these layers added no value on top of linear layers- perhaps due to limited hardware. Kolm et al. [5] show that linear layers also give promising results.
- Hidden Layer Count and Nodes ([3, 4, 5], [1024, 2048, 4096]) → Kolm et al. [5] use four hidden layers, which should be sufficient, and the number of nodes will need to be found through exploration via tuning.
- Learning Rate (0.00001, 0.0001) → Kolm et al. [5] use 0.00001 as their learning rate, so this is used as a baseline.
- Activation function (ReLU, Tanh, Sigmoid) → There is a saturation problem with Tanh and Sigmoid: as the inputs to the functions tend towards positive infinity or negative infinity, the gradient approaches zero, leading to vanishing gradients. ReLU does not have this because it is a linear function when the input is positive. It is also easier to compute ReLU, making it computationally efficient.
- Batch size (128, 256, 512) → Kolm et al. [5] use 256 as their batch size, so this will be the starting point.
- Loss function (MSE, RMSE, MAE) → RMSE (Root mean squared error) is used to make the error more interpretable, but this is not important for training the model. MAE (Mean absolute error) is less sensitive to outliers, but our data is observed from a real data feed, so there are no real outliers. If it is possible to observe a data point that is outside of the expected distribution, then the model should be aware of this. It is common to use MSE (Mean squared error) for training regression models.
- Optimiser method (SGD, ADAM) → ADAM (Adaptive moment estimation) is an extended version of SGD (Stochastic gradient descent) by adapting the learning rate for each parameter based on the gradients from the previous optimisation step. This means that ADAM is less sensitive to the initial learning rate choice as it will continuously change throughout training to a more appropriate value. ADAM also handles different gradient scales better than SGD because it uses the first and second moments of the gradients. Kolm et al. [5] also use ADAM.
- Early Stopping Patience (5, 10) → 5 is recommended by Kolm et al. [5] so this will be the starting point.
- Maximum number of Epochs (50, 100, 150) → 100 is chosen from tolerance.

4.1.2 Methodology for Tuning Parameters

There are many ways to optimise the hyperparameters, but since a reasonable amount of time can be dedicated to tuning, the simplest yet robust method will be implemented: grid search. This involves exhaustively iterating through all defined configurations of hyperparameters and using the combination that has the best validation score. Table 3 contains the proposed configurations for each hyperparameter, and this will be executed for all five instruments. The hyperparameters that will be tuned are the parameters with limited knowledge of their optimal values and, therefore, require experimentation.

Table 3: Supervised Learning Hyperparameter Configurations for Grid Search

Hyperparameter	Configurations
Hidden Layer Count and Nodes	[512] * 4, [1024] * 4, [2048] * 4
Learning Rate	0.00001, 0.0001
Early Stopping Parameter	5, 10
Batch Size	128, 256, 512

There are a total of 36 combinations, and each run takes between 10 and 30 minutes. Taking 20 minutes as an average per run and repeating this for four other instruments leads to a total of 60 hours (20*36*5/60).

4.2 Reinforcement Learning Tuning

Here is a list of all the hyperparameters that will need to be tuned for the reinforcement learning component. As there are three agents, beside each parameter in the list below is/are the agent(s) that it belongs to. Please note that DQN and DDQN will be tuned together. There will be repeated hyperparameters from the supervised learning tuning because there are also neural networks in this component. For such parameters, please refer to the previous Section because the same strategies have also been adopted in this part unless mentioned otherwise.

- (All) Maximum and minimum exploration rate $\rightarrow$ is the probability of making a random action during learning and aims to explore other options, rather than using the learned model, to find better rewards. The exploration rate should be the highest towards the beginning of the learning process and should gradually decrease towards a minimum. If the exploration rate is too low, then the agent will not have a chance to explore the environment enough. If the exploration rate is too high, then the convergence will be slower due to unstable learning, as the agent will be taking too many random/sub-optimal actions.
- (All) Exploration rate decay $\rightarrow$ is the rate at which the exploration rate decays throughout training (exponentially). If the decay decreases the exploration rate too quickly, then the agent will not have enough exposure to learn from alternative decisions. If the decay decreases the exploration too slowly, then the agent may take longer to converge due to unstable learning, as mentioned. The exploration rate at each episode is calculated as ($rate\_decay**episode$) and then set to the minimum exploration rate if exceeded.
- (All) Discounted future reward factor $\rightarrow$ is the importance given to future rewards during learning. A value closer to 0 makes the agent focus on immediate rewards, whereas a value closer to 1 causes the agent to focus on long-term rewards, which encourages more strategic decision-making.
- (All) Number of episodes $\rightarrow$ is the number of times the training set has been iterated. Each iteration through the training set is not the same because the agent will most likely take different actions leading to different rewards and, therefore, learn something new. If this number is too low, then the agent will not learn enough from the environment. Conversely, if this number is too big, then it is very time-consuming and has diminishing returns.
- (All) Reward function $\rightarrow$ is the function that returns a value (reward) when the agent is in a specific state. It is used to teach the agent whenever it does something correctly to get into a desirable state or incorrect to get into an undesirable state and hints the agent on what to prioritise as well as what to ignore.
- (All) Learning rate.
- (Q) State space coverage $\rightarrow$ is the state space covered by the Q table. If the state space coverage is too small, then it is likely to encounter states that are outside this region, which will need to be ignored, and this is not ideal if this occurs often. If the state space covered by the Q table is too large, then there will be redundant space in the Q table that will never be filled- ultimately wasting computational resources.
- (Q) Number of buckets $\rightarrow$ is the precision of separating the state space into buckets. If this value is too low, then it is likely that distant regions in the state space that should be separated will be merged into the same bucket. If this value is too high, then it is computationally expensive, and it will require a lot of time to accurately fill out all the values in the table.
- (DQN, DDQN) Batch size.
- (DQN, DDQN) Experience replay buffer size $\rightarrow$ is the size of the experience replay buffer. If this is too small, then it can cause overfitting to recent data as it can store a limited number of experiences. If this is too large, then it can be computationally expensive to store the data. Furthermore, the buffer is more likely to contain experiences that are outdated because the agent has improved a lot since those experiences were added to the buffer, which leads to redundant learning.
- (DQN, DDQN) Network architecture.
- (DQN, DDQN) Optimiser.
- (DQN, DDQN) Target update frequency $\rightarrow$ corresponds to the number of steps each time the target network updates its parameters to share the parameters of the policy network. If the update frequency is too low, then the target Q values for updating the policy network become outdated,

leading to slow convergence. If the update frequency is too high, then the learning will become unstable because the policy network will try to catch up to a constantly changing target network.

- (DDQN) Target update weight → is the rate at which the target network's weights and biases are updated using the policy network's parameters. If this update weight is too small, then the target network will lag very behind compared to the policy network, causing slow convergence. If the update weight is too large, then the target network is more influenced by the policy network, which can cause learning to become unstable. The policy network will try catching up to a constantly changing target network.

4.2.1 Justifying Values for Certain Parameters

There are certain parameters where their optimal values can be justified using existing knowledge. Hence, they will not need to be tuned unless they are a starting point from existing literature. Here is a list of hyperparameters where their values are justified among popular alternatives.

- (All) Maximum exploration rate (0.8, 0.9, 1.0) → As the initialised model before learning does not contain any relevant information regarding the task, the probability for the agent to execute random actions to learn in the environment should be the maximum possible value, which is 1.
- (All) Minimum exploration rate (0.0, 0.1, 0.2) → After the model has been exposed to the environment for enough episodes, the model should still make random actions to continue learning, but it should be low enough to stabilise learning. A 10% chance for a random action to occur is reasonable when the model is beginning to stabilise.
- (All) Exploration rate decay → The minimum exploration rate should be reached after 70 out of 80 episodes so that it can use the remaining ten episodes to consolidate. Please see the number of episodes parameter below. According to the rate decay equation in the previous subsection, this means that the exploration decay constant must be 0.935 (3 sf).
- (All) Discounted future reward factor (0.9, 0.95, 0.99) → As profits are accumulated in the near future but not too far into the future, 0.99 might take the agent a very long time to converge. After 100 updates to price, the reward will still be contributing roughly 37% to the state action Q value ($0.99^{100} \approx 0.37$). Bertermann [4] uses 0.9, but this will be further explored through tuning.
- (All) Number of episodes (40, 80, 120) → This is chosen from the amount of time waiting for the learning process to finish. This equates to two iterations of the entire training dataset (80 episodes), which takes roughly 45 minutes.
- (All) Reward function (PNL change, Account balance, % profitability) → Each trade should be treated and taken independently. PNL (profit and loss) change is suitable because it returns the reward gained from the previous timestep, which leads to faster convergence. Account balance and % profitability cause dependency issues as they can increase or decrease globally throughout the training process, leading to delayed learning.
- (All) Learning rate (0.1, 0.01, 0.001) → Bertermann [4] uses 0.001, so this will be a starting point.
- (Q) State range covered by Q Table (cover all, cover more, cover less) → If the Q table contains all the state space from the training data, then there will be many cells in the Q table that are empty because of outliers (0.5% of the data). This will waste computational space. The 0.5% of data towards the extremities was removed, which is roughly 2.5 standard deviations from the mean. It is better for an agent to say it does not know than to give an estimate for data points it has not been trained on.
- (Q) Number of buckets per state (3, 5, 7, 9) → This separates each alpha into one of five categories: large bearish bias, small bearish bias, no bias, small bullish bias, and large bullish bias. This should be sufficient information for the Q learning agent to make good decisions. As there are six horizons in the input data, two states for the current trade state, and two possible current actions, the Q table will have $5^6 * 2^2 = 62500$ cells.
- (DQN, DDQN) Batch size (64, 128, 256) → Bertermann [4] did not use experience replay, so there is no starting point. This will have to be trialled through tuning.
- (DQN, DDQN) Experience replay buffer size (4000, 6000, 8000, 10000) → A record of experience should stay in the buffer for 10,000 steps before it is claimed outdated. This is because an agent should make improvements every 10,000 steps, given that there are about 30,000 to 100,000 steps per day in the data.

- (DQN, DDQN) Hidden Layer Count and Nodes ([1, 2, 3], [16, 32, 64]) → The configurations for the hidden layer count and nodes are smaller compared to alpha extraction because classification with two outputs is simpler than regression with six outputs. This will need to be found through tuning.
- (DQN, DDQN) Type of Layers (Linear, LSTM, LSTM-CNN) → The inputs to these networks are much simpler compared to alpha extraction, so the networks will only need to use linear layers.
- (DQN, DDQN) Activation Function (ReLU, Tanh, Sigmoid) → The output layer in these networks has two nodes corresponding to probabilities of buying and selling. As these are probabilities, they need to be between 0 and 1. Sigmoid is a function that returns a value between 0 and 1, so it is suitable as the activation function for the output layer.
- (DQN, DDQN) Optimiser (SGD, ADAM).
- (DQN, DDQN) Target update frequency (1, 5, 10) → Will have to be found through tuning.
- (DDQN) Target update weight (0.1, 0.2, 0.3) → Will have to be found through tuning.

4.2.2 Methodology for Tuning Parameters

Grid search was used to tune the remainder of the hyperparameters. Table 4 contains the proposed configurations for each hyperparameter, and this will be executed for all five instruments. Q Learning and DQN will be tuned separately. Target weight update will be tuned for DDQN while copying the parameters for DQN.

Table 4: Reinforcement Learning Hyperparameter Configurations for Grid Search

Hyperparameter	Configurations
(All) Learning rate	0.01, 0.001
(All) Discounted future reward factor	0.9, 0.95
(DQN, DDQN) Hidden layer count and nodes	[32]*2, [64]*2
(DQN, DDQN) Target update frequency	3, 6
(DQN, DDQN) Batch size	128, 256
(DDQN) Target update weight	0.1, 0.2

There are four combinations for Q Learning, 32 combinations for DQN, and only two combinations for DDQN as it will share the parameters from DQN. It takes about 40 minutes per run, and given that there are five instruments to train on, this will take a total of about 125 hours (40*38*5/60).

5 Evaluation

This Section covers the results of the optimised models, evaluates them using performance metrics, and compares them to a benchmark (random action agent) using statistical significance testing. The models are tested through backtesting with historical held-out test data, and the best models are put through forward testing on the CTrader FXPro [33] platform. This Section concludes with an investigation to try to explain what values cause the agents to reverse their trades.

5.1 Tuning Results

The results from the hyperparameter tuning for the supervised learning and reinforcement learning components will be presented in this Section. This includes showing the best configurations for the parameters for each instrument, the respective learning curves during training, and the results of the held-out test data. Please note that the supervised learning and the reinforcement learning components are separate in this Section but will be combined for backtesting on the same test dataset in the next Section.

5.1.1 Supervised Learning

The supervised learning tuning methodology and table 5 reveals the optimal parameters for each instrument.

Table 5: Best Supervised Learning Hyperparameters from Tuning for Each Instrument

Parameter	XAUUSD, GBPUSD, EURUSD	FTSE100, DE40
Layer/Nodes	[2048]*4	[2048]*4
Learning Rate	**0.0001**	0.00001
Early Stopping Parameter	5	5
Batch Size	256	256

It is reassuring to observe a lot of overlap in the parameters for each instrument because all of these networks are designed to do the same thing. The parameter distinguishing the models is the learning rate, which is ten times larger for XAUUSD, GBPUSD, and EURUSD compared to the indices FTSE100 and DE40. This suggests that XAUUSD, GBPUSD, and EURUSD are more noisy compared to the indices, which is supported by the high number of data points for the three instruments in table 1.

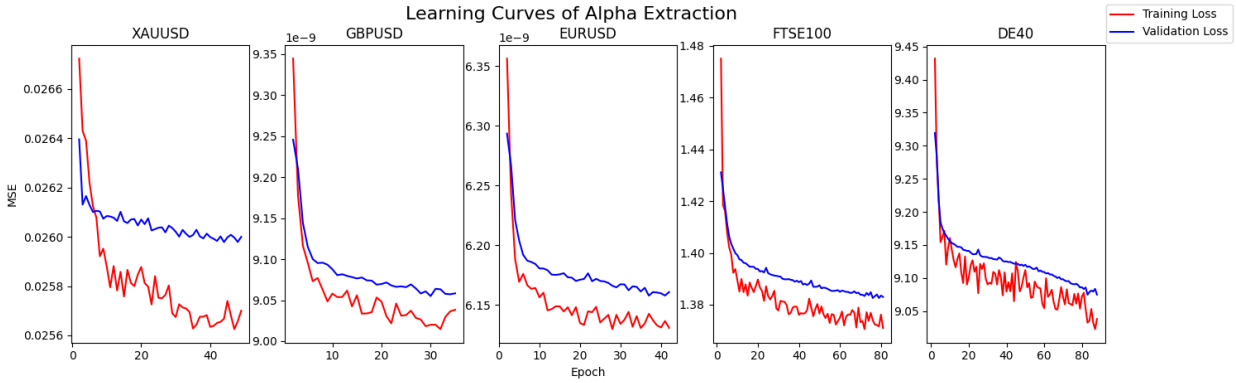


Figure 8: Learning curves of alpha extraction showing training MSE loss (red) and validation loss (blue) for each instrument

Figure 8 reveals the training and validation loss curves during training. The first few epochs are omitted because the losses were too large, and that reduced the detail toward termination. The training curve is quite noisy, but this is to be expected when dealing with market data. Overall, the curves are healthy because the validation loss is close to the training loss, but there are signs of overfitting when the gap is large, such as in XAUUSD (left-most panel). The other models also show overfitting but at a much smaller scale and may be acceptable.

Table 6: Final Training, Validation, and Test Losses for Alpha Extraction (3 s.f.)

Loss (MSE)	XAUUSD	GBPUSD	EURUSD	FTSE100	DE40
Training	0.0257	9.04e-09	6.13e-09	1.37	9.04
Validation	0.0261	9.06e-09	6.16e-09	1.38	9.08
Test	0.0249	9.09e-09	6.19e-06	1.40	9.06

Table 6 presents the results at the end of training, and they are reasonable when compared with each other. For example, the models evaluated on the test set perform as well as on the validation set as expected, and they both are slightly above the training error. However, this does not reveal how good the models are. They need to be compared to a baseline. We use the results from Kolm et al. [5] as our baseline, which requires the breaking down of the model's performance at the horizon level. This is done in table 7, and this presents the RMSE on the test set, the standard deviation of the test set, and the out-of-sample R^2 metric (calculated in equation 14) which is used by Kolm et al. [5] to evaluate their work. There is a theme that the RMSE hovers just under the standard deviation, which is alarming as it indicates that there is a lot of overlap between the predictions and the rest of the data- ranging from 50% to 67% estimated overlap using (2*(pnorm(q=RMSE/std,mean=0,sd=1) - 0.5)). However, the average R^2_{OS} performance matches that observed by Kolm et al. [5] for alpha extraction using an MLP network. They observe an average of 0.181% whereas the results in table 7 show an average of 0.153% and excluding the overfitted XAUUSD model gives an average of 0.180 so there is general confluence between these results.

Table 7: Evaluating Alpha Extraction Error using out-of-sample R^2_{OS} (%) at each Horizon Level

Attribute	Horizon	XAUUSD	GBPUSD	EURUSD	FTSE100	DE40
Test Records	All	523,799	358,196	272,335	187,851	171,758
RMSE	1	0.0386	2.14e-05	1.85e-05	0.348	0.92
Std Dev.	1	0.0313	2.12e-05	1.80e-05	0.317	1.02
R^2_{OS} (%)	1	-0.192	-0.105	-0.113	-0.080	-0.056
RMSE	2	0.0512	3.04e-05	2.52e-05	0.405	0.994
Std Dev.	2	0.0457	3.12e-05	2.60e-05	0.431	1.13
R^2_{OS} (%)	2	-0.009	0.062	0.031	0.094	0.082
RMSE	3	0.0626	3.72e-05	3.10e-05	0.453	1.18
Std Dev.	3	0.0574	3.92e-05	3.23e-05	0.527	1.41
R^2_{OS} (%)	3	0.054	0.153	0.129	0.186	0.173
RMSE	4	0.0680	4.17e-05	3.38e-05	0.502	1.31
Std Dev.	4	0.0673	4.58e-05	3.75e-05	0.605	1.54
R^2_{OS} (%)	4	0.078	0.199	0.201	0.304	0.367
RMSE	5	0.0735	4.64e-05	3.78e-05	0.554	1.40
Std Dev.	5	0.0758	5.16e-05	4.20e-05	0.675	1.73
R^2_{OS} (%)	5	0.152	0.267	0.252	0.348	0.391
RMSE	6	0.0827	4.93e-05	4.10e-05	0.590	1.47
Std Dev.	6	0.0835	5.67e-05	4.61e-05	0.737	1.86
R^2_{OS} (%)	6	0.187	0.320	0.290	0.382	0.428
Average R^2_{OS} (%)	All	**0.045**	**0.149**	**0.132**	**0.206**	**0.231**

Kolm et al. [5] also found that as the fraction of average price change increases (proportional to the horizon), the better the R^2_{OS} (please see figure 5) and table 7 shows this for every instrument. This is perhaps because the higher the horizon, the higher the standard deviation, so the model is more likely to be relatively better compared to the benchmark. There is also agreement that the first horizon is worse than the benchmark used to calculate R^2_{OS}- negative R^2 value shown at horizon level 1. These confluences further support their results despite introducing their work to other asset classes, yet it is interesting to observe that the order in performance from best to worst is DE40, FTSE100, GBPUSD, EURUSD, and XAUUSD. The indices are outperforming the rest of the instruments, and Kolm et al. [5] trained their models on stocks from the NASDAQ index, so equities might be ideal for this setup. Although these results agree with Kolm et al.'s outcomes, the R^2_{OS} values infer that the models are able to explain 0.045% to 0.231% of the variance in alphas. Combining this alpha extraction with the reinforcement learning component will give more insight into the true performance of these supervised learning models.

5.1.2 Reinforcement Learning

The reinforcement learning tuning methodology (see tables 8 and 9 below) shows the optimal parameters for each instrument for each agent. All three agents will be evaluated and compared together.

Table 8: Best Q Learning Hyperparameters from Tuning for Each Instrument

Parameter	XAUUSD	GBPUSD	EURUSD	FTSE100	DE40
Learning rate	0.01	0.01	0.01	0.01	0.01
Discounted future reward factor	**0.9**	0.95	0.95	0.95	0.95

Table 9: Best DQN/DDQN Hyperparameters from Tuning for Each Instrument

Parameter	XAUUSD	GBPUSD	EURUSD	FTSE100	DE40
Learning rate	0.01	0.01	0.01	0.01	0.01
Discounted future reward factor	**0.9**	0.95	0.95	0.95	0.95
Layers/Nodes	[64]*2	[64]*2	[64]*2	[64]*2	[64]*2
Target update frequency	**6**	3	3	3	3
Batch size	128	128	128	128	128
Target update weight	**0.1**	0.2	0.2	0.2	0.2

Once again, it is reassuring to see similar parameter values for each instrument, but there were not many options as shown in table 4. Similar to the supervised learning component, XAUUSD has a different configuration of parameters compared to the other instruments, perhaps due to the noise in its large dataset. This is confirmed by needing to update the target network less often and needing to update the target network parameters less to stabilise learning when compared with other instruments. The discounted future reward factor being lower also infers that alphas do not affect too far into the future due to noise.

Figures 9, 10, and 11 show the reward curves of each agent for each instrument during training. These curves overall indicate difficulty in learning, and this is because the agent needs to learn when it is worth reversing a trade, even though the alphas supplied are as accurate as they can be. The commissions applied to every trade match the costs at CTrader FXPro, which from lowest to highest per lot size is GBPUSD/EURUSD, XAUUSD, FTSE100, and DE40. It is surprising that FTSE100 and DE40 are close to being profitable during training across all agents, while XAUUSD was not profitable at all when having less transaction costs. This shows how poor the quality of the data collected for XAUUSD was given that it has almost all the other total records from the other instruments combined in the same ten weeks. With regards to the foreign exchange pairs having the least commissions, all the agents were very profitable, with Q Learning earning the highest, followed by DDQN, and then DQN- roughly with a ratio of 15:7:5. However, nothing is conclusive as this is only during training.

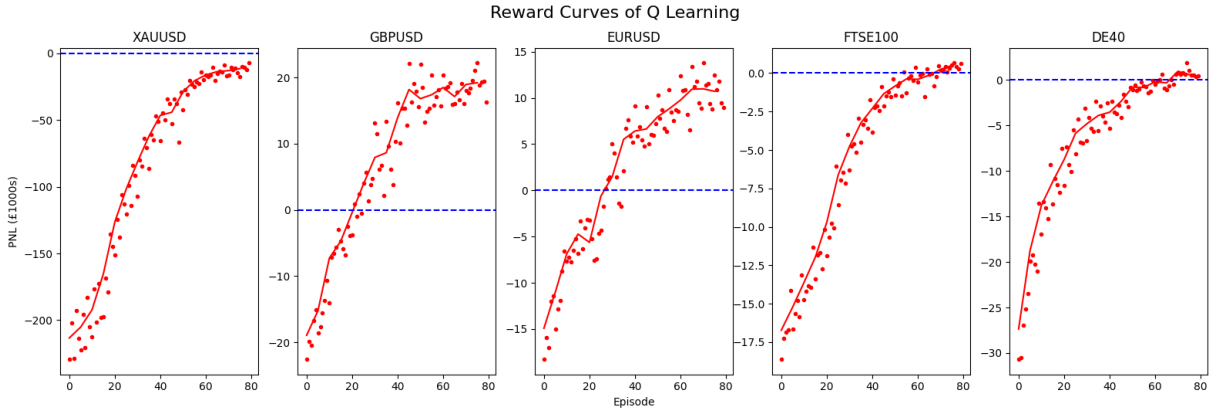


Figure 9: Reward curves of Q Learning for each instrument during training (1 lot size trades)

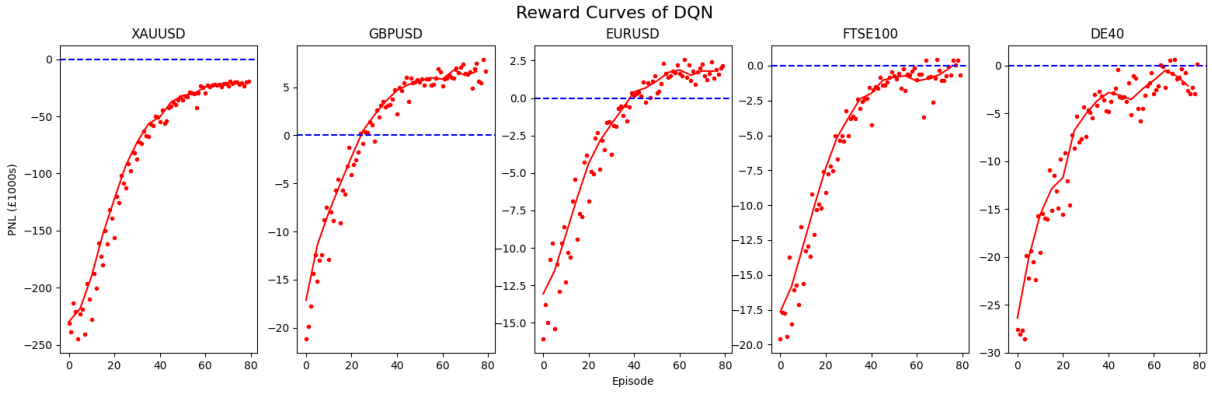


Figure 10: Reward curves of DQN for each instrument during training (1 lot size trades)

Tables 10, 11, and 12 show the calculated performance metrics of each agent on the 5-day test dataset for each instrument. Overall, the results align with the outcome of the reward curves during training, which indicates that these agents generalised well and there is little to no overfitting. Using the daily average profit and volatility, the order of best to worst instruments to apply these models to are GBPUSD, EURUSD, DE40, FTSE100, and XAUUSD.

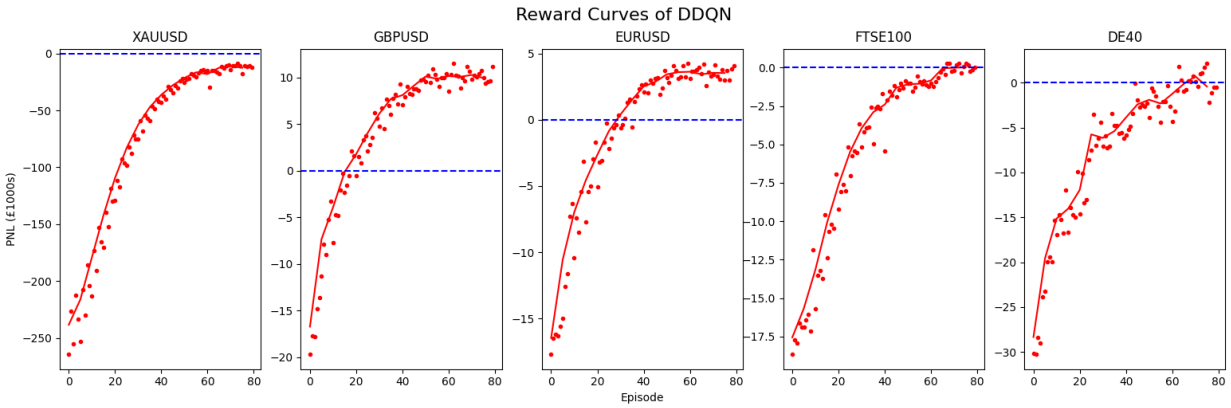


Figure 11: Reward curves of DDQN for each instrument during training (1 lot size trades)

Quantitatively, Q Learning made £8,030 in overall profit, whereas DQN lost £29,800 and DDQN lost £5,460, which is a wide range of outcomes. The row-wise average volatility (standard deviation) for the Q Learning agent (£1820) is almost double the average volatility for the DQN (£1060) and DDQN (£914), with DDQN having the smallest volatility. This can make the neural networks more suitable for stable trading once they become viable. The average profit/loss ratios and the profitability are very close together for DDQN and Q Learning (around 0.83,0.84 ratio with 49,51% profitability) but noticeably worse for DQN (0.77 ratio with 44% profitability).

The ranking of the agents from best to worst is Q Learning, DDQN, and DQN when it comes to overall execution. This supports Bertermann's [4] findings in that Q Learning is generally better than DQN for this environment. Although DDQN is more promising compared to DQN, as expected due to DQN's overestimation bias, Q Learning still outperforms DDQN. This indicates that tabular representation of the states is suitable compared to approximating the Q function as proposed by Bertermann [4] or perhaps more tuning is required to get the most out of the DQN/DDQN. However, it is important to note that Q tables have a limited domain range.

Table 10: Q Learning Performance Metrics on Test Data (3 s.f.)

Metrics	XAUUSD	GBPUSD	EURUSD	FTSE100	DE40
Daily Average Profit (£)	-20,100	18,600	9,890	-252	-107
Daily Average Volatility (£)	3,390	3,280	1,810	274	363
Average Profit/Loss	0.964	1.13	1.01	0.533	0.512
Profitability (%)	35.9	62.8	62.3	43.3	52.4
Maximum Drawdown (£)	-45,200	-1.40	-13.3	-3,260	-1,400

Table 11: DQN Performance Metrics on Test Data (3 s.f.)

Metrics	XAUUSD	GBPUSD	EURUSD	FTSE100	DE40
Daily Average Profit (£)	-32,700	5,410	1,420	-603	-3,300
Daily Average Volatility (£)	1,360	1,560	920	503	946
Average Profit/Loss	0.952	0.983	1.04	0.468	0.417
Profitability (%)	20.1	53.5	54.2	51.6	42.9
Maximum Drawdown (£)	-41,400	-40.1	-1,040	-2,500	-6,020

5.2 Backtesting Results

This Section combines both the alpha extraction (supervised learning component) and the reinforcement learning agents. OFI features will be taken as input to predict alphas using the supervised learning component, which will then be fed into the learning agents to output a trading signal. Tables 13, 14, and 15 showcase the results of the pipeline on the same test data for comparison to tables 10-12.

Table 12: DDQN Performance Metrics on Test Data (3 s.f.)

Metrics	XAUUSD	GBPUSD	EURUSD	FTSE100	DE40
Daily Average Profit (£)	-16,900	8,090	3,410	-55.4	-5.43
Daily Average Volatility (£)	1,430	1,380	905	103	751
Average Profit/Loss	0.931	1.07	1.04	0.570	0.601
Profitability (%)	38.5	57.3	57.9	44.2	51.7
Maximum Drawdown (£)	-35,600	-102	-1,100	-1,040	-3,150

Table 13: Q Learning Performance Metrics on Test Data with Alpha Extraction (3 s.f.)

Metrics	XAUUSD	GBPUSD	EURUSD	FTSE100	DE40
Daily Average Profit (£)	-126,000	2,210	-120	-4,100	-2,990
Daily Average Volatility (£)	9,150	3,590	2,100	593	675
Average Profit/Loss	0.703	0.867	0.841	0.441	0.429
Profitability (%)	23.6	53.6	53.5	31.7	38.3
Maximum Drawdown (£)	-462,000	-11,800	-9,510	-15,100	-10,700

Overall, there is a noticeable drop in performance, which is expected because the alpha extraction is not the most accurate, while the learning agents were trained on accurate alphas. There is about a £100,000 decrease in daily average profit for gold, £15,000 decrease for the foreign exchange pairs, and £3,000 decrease for the indices- confirming the quality of the alpha extraction order regarding instruments already discussed. This brings all the profits from positive to negative except Q Learning on GBPUSD, which performs at a daily average of £2,210 with a standard deviation of £3,590. Q Learning on EURUSD follows in 2nd place at a daily average of -£120 with a standard deviation of £2,100, which shows potential as some days are profitable. The standard deviation has also increased in the range of 20% to 200%, increasing uncertainty as expected. The other metrics also indicate slightly poorer performance. All in all, Q Learning still outperforms DDQN, and DDQN outperforms DQN, matching the trend prior to incorporating the alpha extraction models.

Please note that the performance of these models might vary for another set of five days of test data, and so more data is required to make concrete conclusions about profitability. The analysis conducted in these sections should only be taken as an indication of performance.

5.2.1 Benchmarking and Statistical Significance

Although the backtesting results generally reveal poor performance apart from potential profitability with GBPUSD and EURUSD using Q Learning (further discussed in forward testing results), the outcome can be compared to the metrics of an agent that executes random actions on the same test dataset, and this is shown in table 16.

Executing on the same dataset allows for fair comparison, and statistical significance testing can be conducted to provide evidence, for formality, that the models perform better than the random agent. Due to there only being 5 data points (5 days of testing), it is better to use a non-parametric test such as the Mann-Whitney U-test (explained in [39]). The daily profit from the trained agents and the random agent from backtesting on the test data will be used as input to the U-test. The one-tailed null hypothesis is that the trained models do not produce more profit compared to the random agent, and so the U value associated with each trained needs to be calculated and compared to the critical value in order to reject the null hypothesis.

Table 14: DQN Performance Metrics on Test Data with Alpha Extraction(3 s.f.)

Metrics	XAUUSD	GBPUSD	EURUSD	FTSE100	DE40
Daily Average Profit (£)	-151,000	-11,500	-12,000	-4,830	-6,750
Daily Average Volatility (£)	6,520	1,840	1,530	926	1,380
Average Profit/Loss	0.684	0.766	0.784	0.410	0.392
Profitability (%)	19.1	43.6	44.9	42.2	39.6
Maximum Drawdown (£)	-337,000	-33,200	-38,000	-27,600	-29,900

Table 15: DDQN Performance Metrics on Test Data with Alpha Extraction (3 s.f.)

Metrics	XAUUSD	GBPUSD	EURUSD	FTSE100	DE40
Daily Average Profit (£)	-114,000	-8,290	-10,500	-3,910	-3,010
Daily Average Volatility (£)	6,780	2,150	1,300	491	1,120
Average Profit/Loss	0.719	0.855	0.846	0.513	0.549
Profitability (%)	25.6	50.2	51.4	38.3	45.9
Maximum Drawdown (£)	-241,000	-28,800	-35,300	-29,100	-19,800

Table 16: Random Action Performance Metrics on Test Data (3 s.f.)

Metrics	XAUUSD	GBPUSD	EURUSD	FTSE100	DE40
Daily Average Profit (£)	-236,000	-21,600	-16,600	-18,700	-10,300
Daily Average Volatility (£)	18,900	1,820	2,420	2,060	1,200
Average Profit/Loss	0.432	0.739	0.697	0.254	0.558
Profitability (%)	18.5	43.2	42.8	12.6	40.1
Maximum Drawdown (£)	-1,180,000	-108,000	-82,900	-93,500	-51,000

The steps in [39] were followed. All the sums of ranks for the Mann-Whitney U-test are 40, meaning that the profits from the trained models were all more than the profits from the random agent per day on the test dataset. Therefore, the process is the same for all the agents. The U value associated with each trained agent (for higher values) is 0. As ranks are always positive and the U value associated with each trained agent is 0, the null hypothesis is always rejected at all significant levels. Hence, the trained models are better than the random agent benchmark.

5.3 Forward Testing Results

This Section takes the agents performing well on certain instruments during backtesting and puts them in forward testing environments for real-time trading on the CTrader FXPro platform. As mentioned, this involves hosting a web application that will load the agents and wait for POST requests from the platform. The platform will send OFI data and retrieve an action, which will be executed on the platform.

The best-performing agents were selected based on the average daily profit being close to positive, and this was Q Learning on both GBPUSD and EURUSD. These will be the contenders to the forward testing pipeline. Forward testing was only conducted for an hour each, so due to the low sample size, the results can vary. Table 12 shows the profit and loss graphs for Q Learning on GBPUSD and EURUSD.

Overall, it shows negative results with earning -£2,400 for GBPUSD and -£2,640 (£240 lower) for EURUSD, but there are a couple of reasons for this poor outcome. Firstly, the sample size is too small, and it could be profitable over longer periods of time, so more data is required to make a better judgement on its performance. This is less likely, though. Secondly and more importantly, the trading bot often skips a few timesteps while it processes the OFI values to return an action. During backtesting, the environment would wait for the agent to make an action before moving to the next step, whereas during forward testing, the environment does not wait and keeps moving, and then the trade is executed too late. Better hardware is required to mitigate this, and alternate solutions should be explored to replace the web application and have the agent be directly integrated with the platform to speed up the process. This was not possible from initial research.

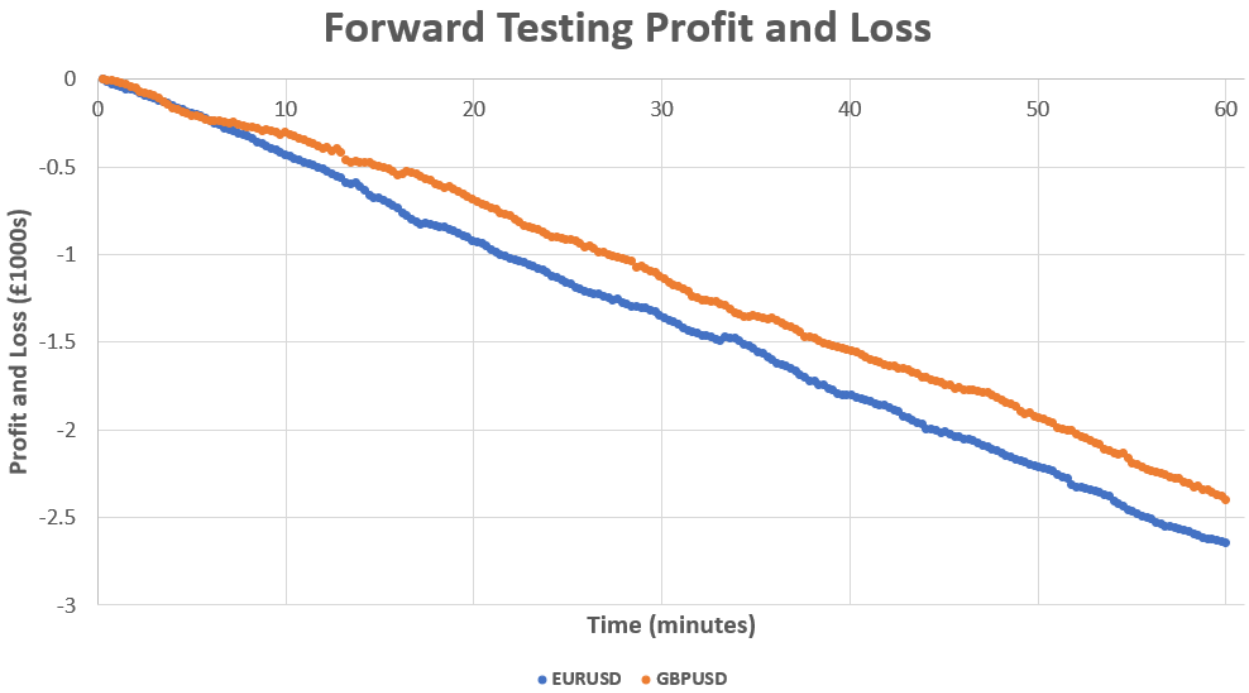


Figure 12: Forward testing results using Q Learning on GBPUSD and EURUSD

5.4 Explainable AI

This Section attempts to explain the decisions made by the agents and investigates what horizon levels contribute the most to making the decision to send a reversal signal from short to long and short to long. Figures 13, 14, and 15 show heatmaps representing the average approximated Q values normalised across all instruments for each agent. DQN and DDQN are shown as tabular approximations by evaluating their functions at intervals equivalent to the bucket size for the Q Learning table to ensure a fair comparison. The state space (x-axis) contains 99.5% of possible alpha values from the training dataset, which is 2.5 standard deviations from either side of the mean (close to 0).

Overall, these heatmaps show expected trends, such as having high positive Q values at positive alphas when choosing to go from short to long, and vice versa, with having high positive Q values at negative alphas when choosing to go from long to short. It is interesting to see that all agents have negative Q values in the neutral column, indicating that it is better not to reverse trades near alpha values of 0 (neutral), i.e. to avoid transaction costs when price will not move at all. This is expected, but the Q values are slightly positive at negative alphas when choosing to go from short to long and vice versa, which indicates a flaw in logic as it chooses not to trade at neutral alpha values at all. This is interesting because, in theory, it suggests that the agent is purposely trying to lose money, but in practice, the agent probably received enough reward whenever it did this, which paid off when the trade duration elapsed past the horizon window and worked out.

Although all the horizons are used to ultimately make the decision of whether to reverse the trade, the first horizon is predominantly used for Q Learning to both long and short, DQN to long, and DDQN to long and short, whereas the third horizon is used by DQN to short. The fifth and sixth horizons are used to long by DQN and DDQN, respectfully. However, all in all, the first horizon is mainly used to make decisions. This further supports why forward testing produced poor results, as the first horizon is almost always elapsed by the time a new trade was executed due to the relatively long process time of making predictions.

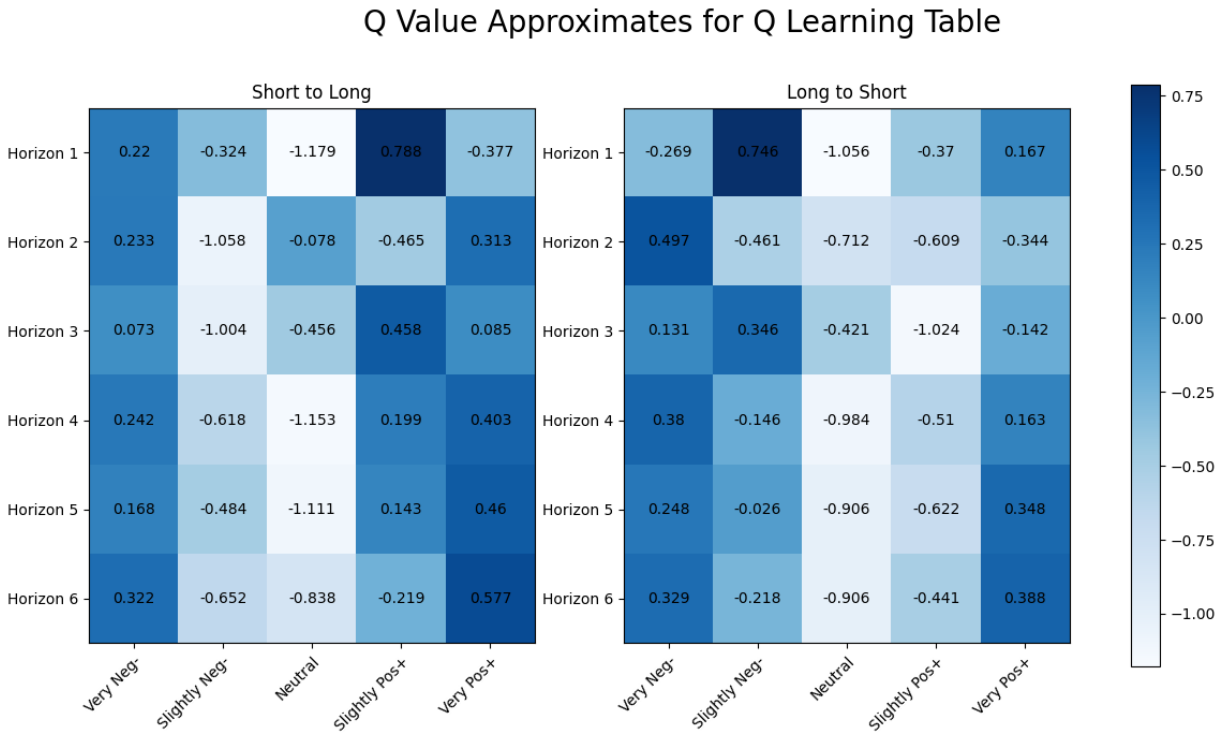


Figure 13: Q value approximates for Q Learning table (averaged across all instruments)

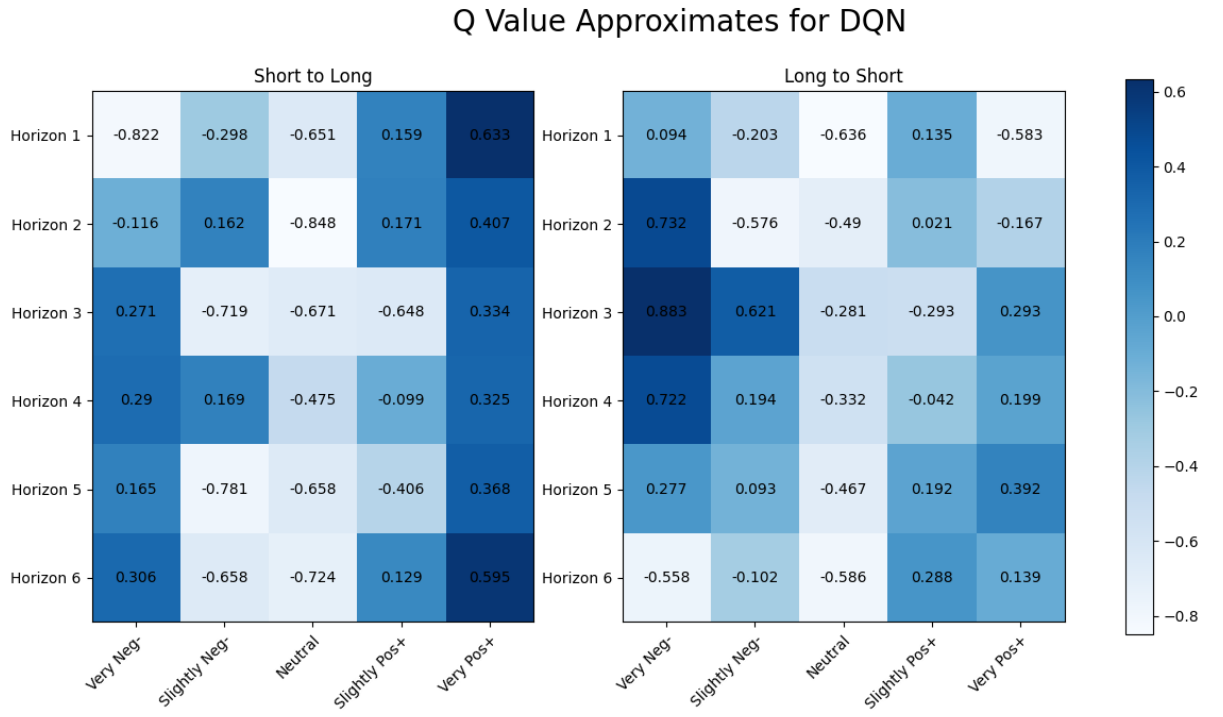


Figure 14: Q value approximates for DQN (averaged across all instruments)

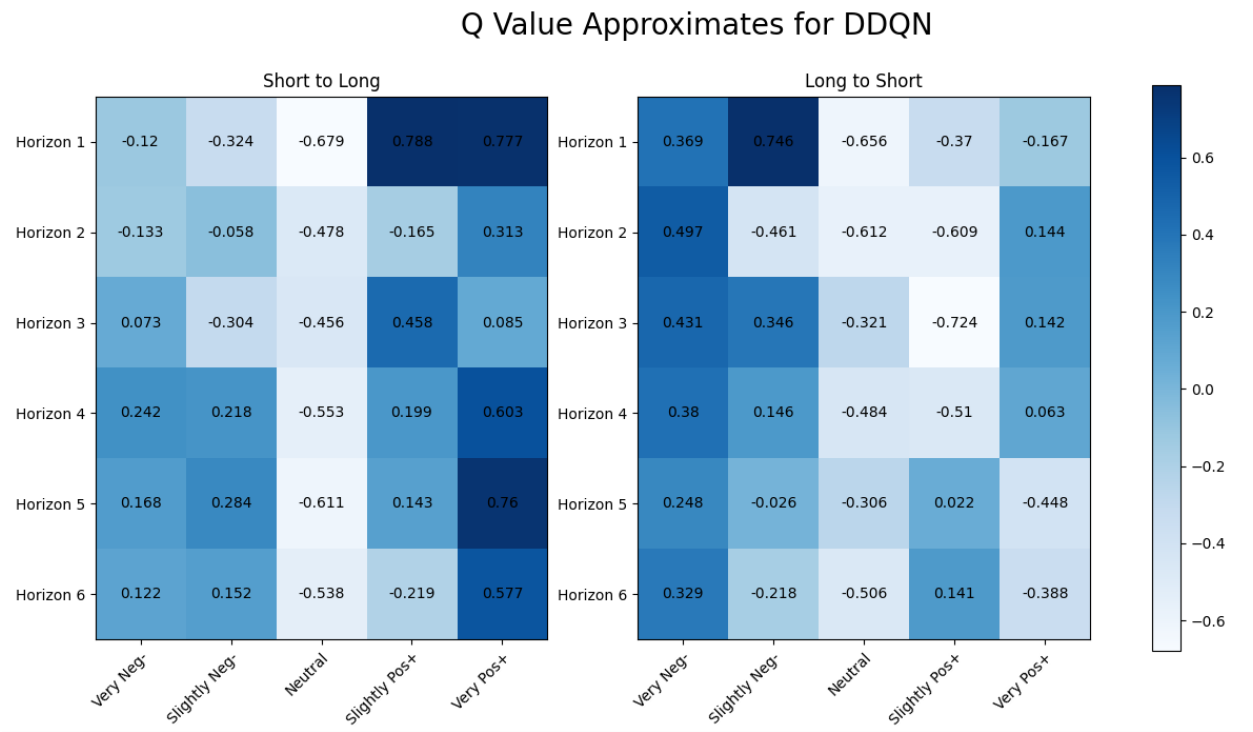


Figure 15: Q value approximates for DDQN (averaged across all instruments)

6 Conclusions and Further Work

This research sets out to combine deep learning on the order books with reinforcement learning to break down large-scale end-to-end models into more manageable and lightweight components for reproducibility, suitable for retail trading. An alpha extraction model forecasts return over the next six timesteps, and a temporal-difference agent will use these values to provide trading signals (buy or sell). One alpha extraction model and three different temporal-difference agents were trained for five financial instruments, giving a total of 20 models and 15 trading bots.

The results for the different components align with the related literature. The alpha extraction outcome aligns with Kolm et al.'s [5] results using the MLP architecture, and the rankings amongst the agents align with Bertermann's [4] findings. Only ten weeks of order flow imbalance data were collected for each instrument and split into 8:1:1 for training, validation, and testing. Grid search was used to find optimal parameters for each model, and more likely than not, values suggested in the literature were found to be best.

Overall, backtesting with retail costs produced promising results, with profitability achieved using Q Learning on GBPUSD and EURUSD, but failed to bring similar results during forward testing. This is due to long processing times of making predictions, sometimes skipping two timesteps, but this can be mitigated with a different infrastructure to using web applications as well as with more advanced hardware. The current setup uses Intel i9-9900K CPU @ 3.60GHz 16GB and Nvidia GeForce RTX 2080 Super 16GB.

The agents, on top of learning expected patterns, also learned patterns that were undesirable due to exposure to receiving positive rewards beyond the horizon window despite observing contradictory forecasting values. This can be solved by increasing this horizon window to beyond six. Other improvements consist of collecting more data, using better hardware, using rate of return instead to standardise across multiple instruments, using another trading platform with lower commissions, and expanding the action space to also not be in any position (i.e. three actions of buy, sell, and not be in a position instead of just buy and sell).

With regard to legal, social, ethical, and professional considerations, the only concern is scraping raw limit order book data from the CTrader platform and storing it locally, which raises ethical issues. Scraping is not a legal violation of their EULA [40] as this work is for research and not commercial gain. This issue was minimised by only storing wanted order flow imbalance features inferred from the raw limit order book states instead of storing actual raw states directly.

A Supplementary Figures

Q Learning	$\overline{PnL}$	σ_{PnL}	Investments/Episode	Profit/Investment
Episodes 0-100	0.1671	0.4839	49.46	0.0034
Episodes 400-500	1.2840	0.4183	46.51	0.0276
Episodes 1400-1500	2.0687	0.4418	39.61	0.0522
Episodes 2000-2100	2.1129	0.3542	39.09	0.0541
Episodes 2400-2500	2.1381	0.3956	38.64	0.0553

DQN	$\overline{PnL}$	σ_{PnL}	Investments/Episode	Profit/Investment
Episodes 0-100	-0.0061	0.4916	49.15	-0.0001
Episodes 400-500	0.6417	0.5354	38.74	0.0166
Episodes 1400-1500	1.1383	0.7223	21.89	0.0520
Episodes 2000-2100	1.4255	0.6826	21.79	0.0654
Episodes 2400-2500	1.5758	0.6206	20.98	0.0751

Figure 16: Tabular performance comparison of Q Learning and DQN provided by Bertermann [4, Page 29]

Table 17: Mean of Values at each OFI Level from the Collected Data (4 d.p.)

OFI Level	XAUUSD	GBPUSD	EURUSD	FTSE100	DE40
1	0.1236	0.0054	0.0015	0.0251	0.0350
2	0.1032	0.0312	0.0303	0.1119	0.0073
3	0.1241	0.0567	0.0569	0.1299	0.0122
4	0.1488	0.1037	0.1016	0.1334	0.0175
5	0.1763	0.1470	0.1510	0.1780	0.0226
6	0.2043	0.1679	0.1313	0.2077	0.0564
7	0.0566	0.0875	0.0489	0.1057	0.0113
8	0.0779	0.1190	0.0913	0.0818	0.0097
9	0.0956	0.0828	0.1341	0.0456	0.0143
10	0.1078	0.0445	0.0282	0.0332	0.0163

Table 18: Standard deviation of Values at each OFI Level from the Collected Data (4 d.p.)

OFI Level	XAUUSD	GBPUSD	EURUSD	FTSE100	DE40
1	0.2814	0.1267	0.0774	0.3985	0.9989
2	0.3444	0.2261	0.2730	0.6038	0.1844
3	0.3962	0.3301	0.3995	0.4512	0.1985
4	0.4553	0.4233	0.4646	0.4819	0.2021
5	0.5133	0.4910	0.4674	0.5603	0.2149
6	0.5674	0.5434	0.4190	0.4896	0.2553
7	0.3273	0.4199	0.3591	0.4857	0.1846
8	0.3775	0.4572	0.4729	0.5083	0.2177
9	0.4016	0.3837	0.5153	0.4034	0.2152
10	0.4302	0.4472	0.4673	0.3846	0.1892

Table 19: Percentage of Positive Entries at each OFI Level from the Collected Data (1 d.p.)

OFI Level	XAUUSD	GBPUSD	EURUSD	FTSE100	DE40
1	52.8	53.9	53.3	53.1	51.7
2	66.1	62.2	60.5	55.8	53.1
3	67.0	62.8	60.2	53.5	57.5
4	67.2	64.0	61.9	49.1	56.9
5	67.8	64.5	65.2	46.6	56.2
6	67.8	62.6	61.7	35.3	52.7
7	25.1	35.6	36.7	24.6	21.9
8	24.3	31.2	35.4	21.6	21.0
9	22.5	21.2	30.1	19.2	20.9
10	18.9	15.3	15.0	17.5	21.2

Table 20: Percentage of Negative Entries at each OFI Level from the Collected Data (1 d.p.)

OFI Level	XAUUSD	GBPUSD	EURUSD	FTSE100	DE40
1	19.6	32.3	34.5	46.9	48.2
2	28.2	37.0	39.2	43.0	46.6
3	28.2	36.4	39.6	29.6	42.1
4	28.9	34.2	37.5	24.2	41.2
5	29.3	32.7	33.4	23.3	40.4
6	29.4	32.0	29.2	10.	32.1
7	12.3	19.6	22.8	12.4	18.9
8	11.6	15.5	20.9	14.9	16.9
9	9.8	10.6	15.5	15.4	17.7
10	7.1	9.3	10.8	15.0	16.7

Table 21: Percentage of Zero Entries at each OFI Level from the Collected Data (1 d.p.)

OFI Level	XAUUSD	GBPUSD	EURUSD	FTSE100	DE40
1	27.6	13.8	12.2	0.0	0.1
2	5.7	0.9	0.3	1.1	0.3
3	4.8	0.9	0.2	16.8	0.4
4	3.9	1.8	0.7	26.6	1.9
5	2.9	2.9	1.4	30.1	3.4
6	2.7	5.4	9.1	54.7	15.2
7	62.6	44.8	40.5	63.0	59.3
8	64.1	53.2	43.6	63.5	62.1
9	67.7	68.2	54.4	65.4	61.3
10	74.0	75.3	74.2	67.5	62.1

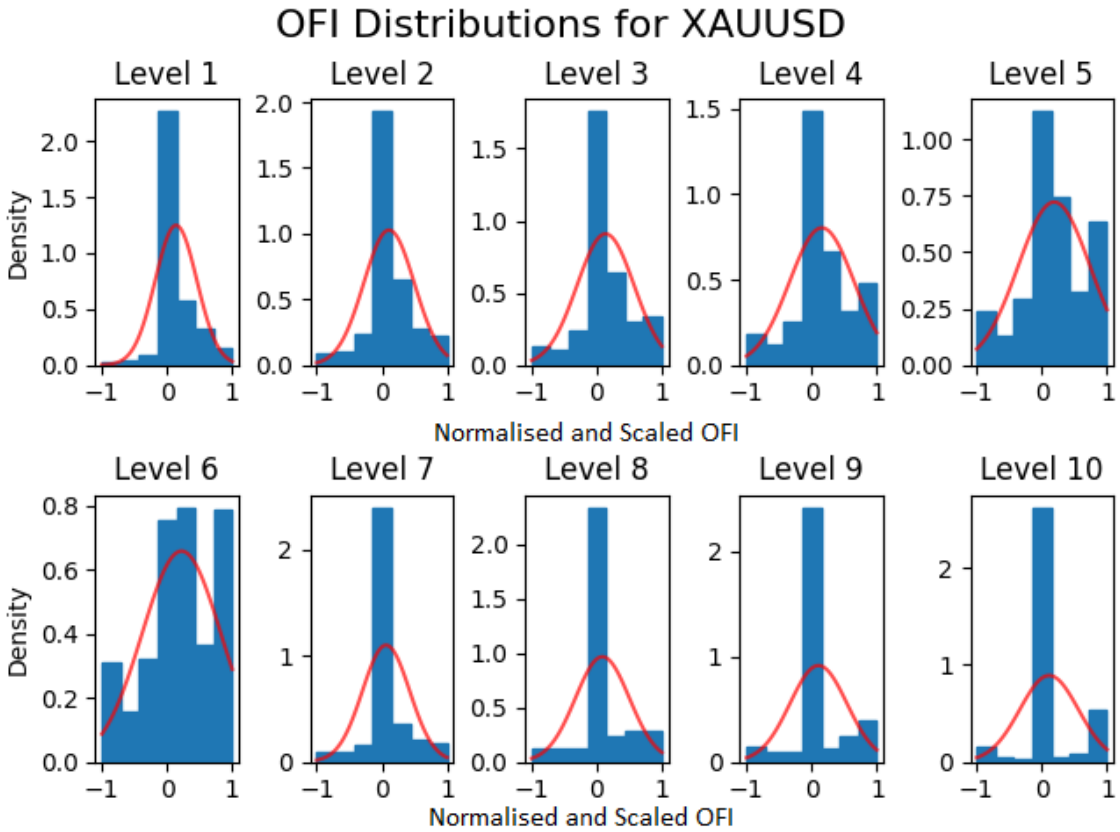


Figure 17: Distribution of first ten OFI levels (blue) from collected XAUUSD (gold) data against a fitted normal distribution (red)

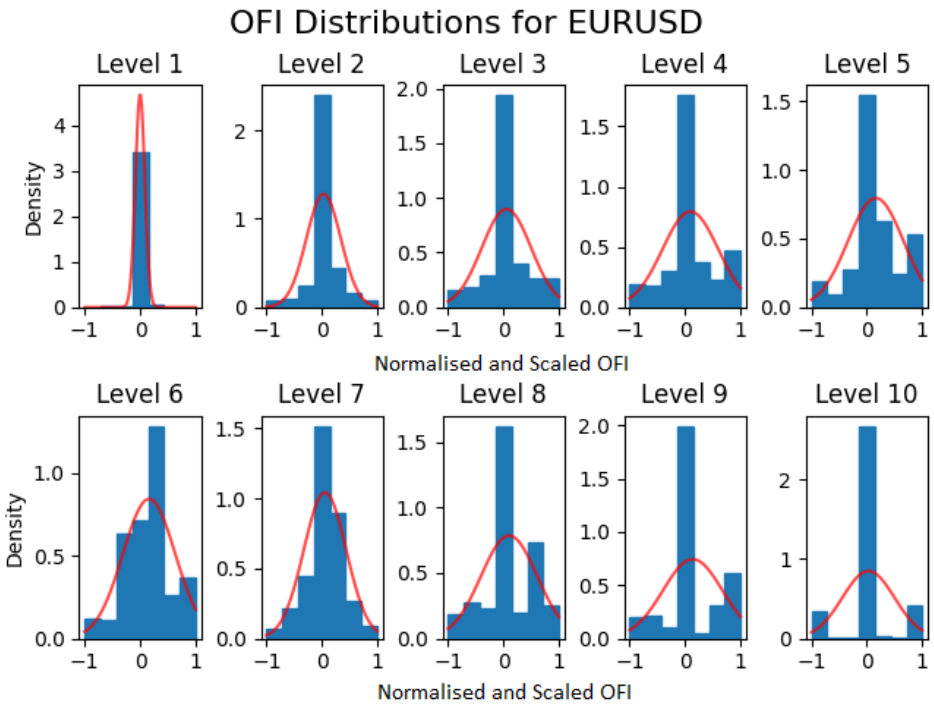


Figure 18: Distribution of first ten OFI levels (blue) from collected EURUSD data against a fitted normal distribution (red)

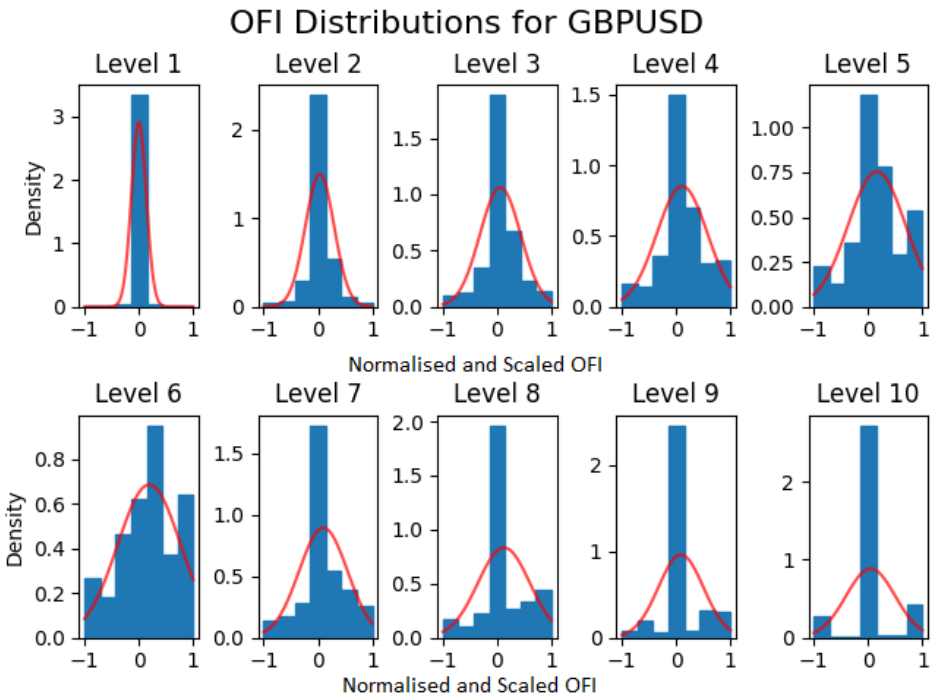


Figure 19: Distribution of first ten OFI levels (blue) from collected GBPUSD data against a fitted normal distribution (red)

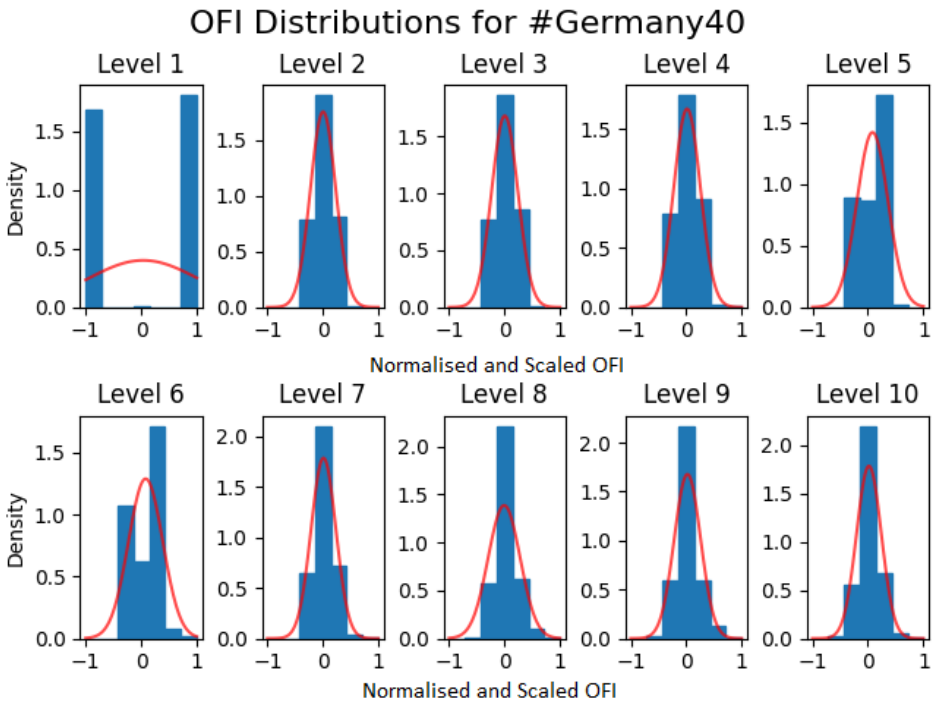


Figure 20: Distribution of first ten OFI levels (blue) from collected DE40 data against a fitted normal distribution (red)

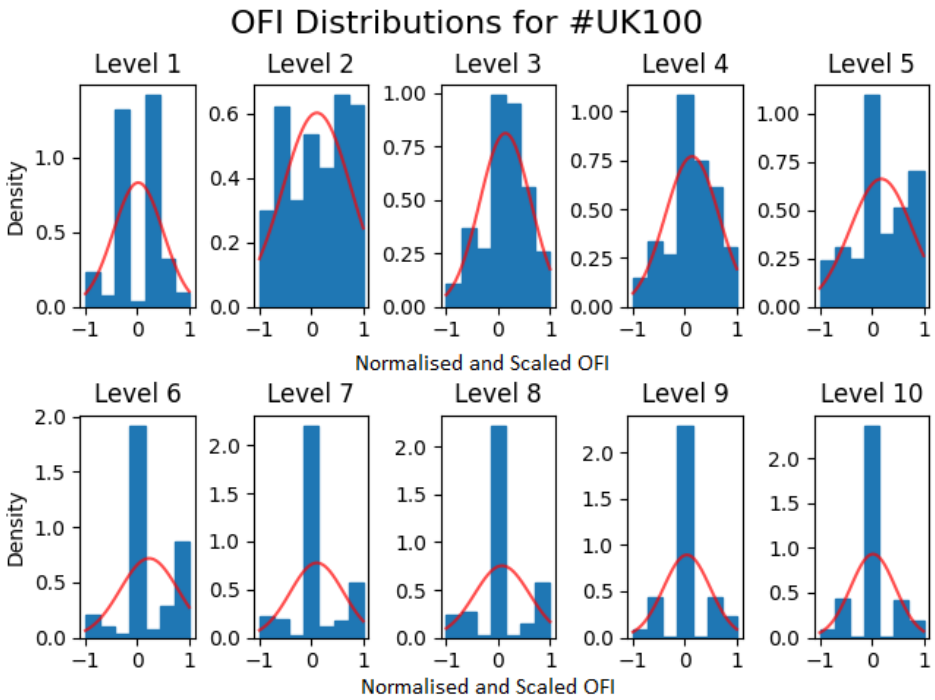


Figure 21: Distribution of first ten OFI levels (blue) from collected FTSE100 data against a fitted normal distribution (red)

Table 22: Mean of Alphas (in Pips) at each Horizon from Collected Data (3 s.f.)

Horizon	XAUUSD	GBPUSD	EURUSD	FTSE100	DE40
1	-8.11e-04	3.37e-05	-1.12e-05	-4.09e-03	-5.07e-03
2	-1.63e-03	6.69e-05	-2.35e-05	-8.15e-03	-1.02e-02
3	-2.44e-03	9.96e-05	-3.62e-05	-1.22e-02	-1.53e-02
4	-3.26e-03	1.31e-04	-4.87e-05	-1.63e-02	-2.05e-02
5	-4.08e-03	1.63e-04	-6.14e-05	-2.03e-02	-2.56e-02
6	-4.90e-03	1.95e-04	-7.40e-05	-2.44e-02	-3.07e-02

Table 23: Standard Deviation of Alphas (in Pips) at each Horizon from Collected Data (3 s.f.)

Horizon	XAUUSD	GBPUSD	EURUSD	FTSE100	DE40
1	2.36	0.155	0.138	3.10	10.1
2	3.48	0.230	0.200	4.08	10.7
3	4.38	0.289	0.251	4.94	13.5
4	5.14	0.339	0.293	5.64	14.6
5	5.81	0.383	0.330	6.28	16.4
6	6.41	0.423	0.363	6.84	17.5

Table 24: Percentage of Positive Alphas at each Horizon from Collected Data (1 d.p.)

Horizon	XAUUSD	GBPUSD	EURUSD	FTSE100	DE40
1	28.8	41.3	39.9	46.6	48.2
2	32.7	30.0	28.7	28.3	16.3
3	38.0	41.5	39.8	44.4	47.8
4	39.8	36.9	35.5	35.0	23.5
5	41.8	42.4	40.8	44.2	47.5
6	42.8	40.0	38.8	38.6	28.0

Time	OFI	Mid Price	Alpha 1	Alpha 2	Alpha 3	Alpha 4	Alpha 5	Alpha 6
23/5/2023	[1.25, 27.5, 97, 95, 67.5, 85, 0, 0, 0, 0]	1.08106	5.00E-06	3.00E-05	4.00E-05	3.00E-05	4.00E-05	3.00E-05
23/5/2023	[2.5, 53.5, 57, 115, 50, 102.5, 0, 0, 0, 0]	1.081065	2.50E-05	3.50E-05	2.50E-05	3.50E-05	2.50E-05	3.50E-05
23/5/2023	[2.5, 23.84, 37, 110, 27.5, 155.5, 0, 0, 0, 0]	1.08109	1.00E-05	0	1.00E-05	0	1.00E-05	5.00E-06
23/5/2023	[2.5, 50, 50, 92.5, 102.5, 71.5, 0, 0, 0, 0]	1.0811	-1.00E-05	0	-1.00E-05	0	-5.00E-06	1.50E-05
23/5/2023	[-1.25, -8, -15, -32, -90, -23.5, 0, 0, 0, 0]	1.08109	1.00E-05	0	1.00E-05	5.00E-06	2.50E-05	2.00E-05
23/5/2023	[2.5, -2.875, 57.5, 72.5, 25, 20.5, 0, 0, 0, 0]	1.0811	-1.00E-05	0	-5.00E-06	1.50E-05	1.00E-05	1.50E-05
23/5/2023	[-1.25, -12, -64.5, 0, 57.5, -8, 0, 0, 0, 0]	1.08109	1.00E-05	5.00E-06	2.50E-05	2.00E-05	2.50E-05	0
23/5/2023	[2.5, 32, 52.5, 67.5, 48, 97, 0, 0, 0, 0]	1.0811	-5.00E-06	1.50E-05	1.00E-05	1.50E-05	-1.00E-05	-5.00E-06
23/5/2023	[0, 27, 44.5, 35.5, -31.5, -36.5, 0, 0, 0, 0]	1.081095	2.00E-05	1.50E-05	2.00E-05	-5.00E-06	0	-5.00E-06
23/5/2023	[2.5, 59.625, 62, 115.5, 115, 54.5, 0, 0, 0, 0]	1.081115	-5.00E-06	0	-2.50E-05	-2.00E-05	-2.50E-05	-2.00E-05

Figure 22: First ten records of the CSV file containing the alpha data calculated from the collected data

Table 25: Percentage of Negative Alphas at each Horizon from Collected Data (1 d.p.)

Horizon	XAUUSD	GBPUSD	EURUSD	FTSE100	DE40
1	29.4	41.2	39.8	46.9	48.2
2	33.2	30.0	28.8	28.2	16.4
3	38.6	41.5	39.9	44.4	47.7
4	40.4	36.8	35.7	35.0	23.5
5	42.5	42.3	41.0	44.1	47.4
6	43.4	40.0	39.0	38.5	28.0

Table 26: Percentage of Zero Alphas at each Horizon from Collected Data (1 d.p.)

Horizon	XAUUSD	GBPUSD	EURUSD	FTSE100	DE40
1	41.8	17.5	20.3	6.5	3.6
2	34.1	40.0	42.5	43.5	67.3
3	23.4	17.0	20.3	11.2	4.5
4	19.8	26.3	28.8	30.0	53.0
5	15.7	15.3	18.2	11.7	5.1
6	13.8	20.0	22.2	22.9	44.0

References

[1] Jennifer Wu, Michael Siegel and Joshua Manion
Online Trading: An Internet Revolution
https://web.mit.edu/smadnick/www/wp2/2000-02-SWP%234104.pdf
MIT, Cambridge, MA, June 1999
pages

[2] Johannes Breckenfelder
How does competition among high-frequency traders affect market liquidity?
https://www.ecb.europa.eu/pub/economic-research/resbull/2020/html/ecb.rb201215~210477c6b0.en.pdf, European Central Bank, December 2020
pages

[3] Salim Lahmiri and Stelios Bekiros
Deep Learning Forecasting in Cryptocurrency High-Frequency Trading
https://link.springer.com/article/10.1007/s12559-021-09841-w
Springer, February 2021
pages

[4] Arvind Bertermann
Reinforcement Learning Trading Strategies with Limit Orders and High Frequency Signals, Imperial College London, September 2021
pages

[5] Petter N. Kolm, Jeremy Turiel and Nicholas Westray
Deep Order Flow Imbalance: Extracting Alpha at Multiple Horizons from the Limit Order Book, https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3900141
SSRN, August 2021
pages

[6] Solveig Badillo, Balazs Banfai, Fabian Birzele, Iakov I. Davydov, Lucy Hutchinson, Tony Kam-Thong, Juliane Siebourg-Polster, Bernhard Steiert, and Jitao David Zhang, *An Introduction to Machine Learning*
https://www.researchgate.net/publication/339680577_An_Introduction_to_Machine_Learning, Clinical Pharmacology & Therapeutics, Janurary 2020
pages

[7] Rian Dolphin, *LSTM Networks | A Detailed Explanation*
https://towardsdatascience.com/lstm-networks-a-detailed-explanation-8fae6aefc7f9, Medium Blogs, October 2020
pages

[8] Jonathan Johnson, *Top Machine Learning Architectures Explained*
https://www.bmc.com/blogs/machine-learning-architecture/
BMC Blogs, September 2020
pages

[9] Simeon Kostadinov, *Understanding Backpropagation Algorithm*
https://towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd, Medium Blogs, August 2019
pages

[10] Jonathon Johnson, *What's a Deep Network? Deep Nets Explained*
https://www.bmc.com/blogs/deep-neural-network
BMC Blogs, July 2020
pages

[11] Aditi Mittal, *Understanding RNN and LSTM*
https://aditi-mittal.medium.com/understanding-rnn-and-lstm-f7cdf6dfc14e
Medium Blogs, October 2019
pages

[12] Richard S. Sutton, Andrew G. Barto *Reinforcement Learning: An Introduction*
The MIT Press, Cambridge, MA, 2018
pages

[13] Reza Jafari, M.M. Javidi, Marjan Kuchaki Rafsanjani
Using deep reinforcement learning approach for solving the multiple sequence alignment problem , Springer, June 2019
pages

[14] Hado van Hasselt, Arthur Guez, and David Silver,
Deep Reinforcement Learning with Double Q-Learningn,
Proceedings of the AAAI conference on artificial intelligence,
https://ojs.aaai.org/index.php/AAAI/article/view/10295, 2016
pages

[15] Ben Hambly, *Introduction to Limit Order Book Markets*
https://www.maths.ox.ac.uk/system/files/attachments/NUS-LOB19.pdf
Oxford, 2019
pages

[16] Mario Köppen *The curse of dimensionality* In 5th online world conference on soft computing in industrial applications (Vol. 1, pp. 4-8)
https://www.class-specific.com/csf/papers/hidim.pdf, Sep 2000
pages

[17] CMC Markets, *Mean Reversion Trading Strategies*,
https://www.cmcmarkets.com/en-gb/trading-guides/mean-reversion, 2023
pages

[18] CMC Markets, *Momentum Trading Strategies*,
https://www.cmcmarkets.com/en-gb/trading-guides/momentum-trading, 2023
pages

[19] Jaroslav Kohout, *Volume Delta Reversal Trade Strategy*,
https://axiafutures.com/blog/volume-delta-reversal-trade-strategy/, 2022
pages

[20] Rama Cont, Arseniy Kukanov, and Sasha Stoikov
The Price Impact Of Order Book Events
Journal Of Financial Econometrics 12.1, 2014
pages

[21] Ymir Mäkinen, Juho Kanniainen, Moncef Gabbouj, and Alexandros Iosifidis, *Forecasting Jump Arrivals In Stock Prices: New Attention-Based Network Architecture Using Limit Order Book Data*, Quantitative Finance, 2019
pages

[22] Nikolaos Passalis, Anastasios Tefas, Juho Kanniainen, Moncef Gabbouj, and Alexandros Iosifidis, *Temporal Logistic Neural Bag-Of-Features For Financial Time Series Forecasting Leveraging Limit Order Book Data*
Pattern Recognition Letters, 2020
pages

[23] Dat Thanh Tran, Alexandros Iosifidis, Juho Kanniainen, and Moncef Gabbouj
Temporal Attention-Augmented Bilinear Network For Financial Time-Series Data Analysis, IEEE Transactions On Neural Networks And Learning Systems, 2019
pages

[24] Ruihong Huang and Tomas Polak,
LOBSTER: Limit Order Book Reconstruction System
https://papers.ssrn.com/sol3/papers.cfm?abstract_id=1977207, 2011
pages

[25] ByBitHelp, *Introduction to TWAP Strategy*
https://www.bybithelp.com/en-US/s/article/Introduction-to-TWAP-Strategys, August 2023
pages

[26] Michaël Karpe, Jin Fang, Zhongyao Ma, and Chen Wang,
Multi-agent reinforcement learning in a realistic limit order book market simulation,
Proceedings of the First ACM International Conference on AI in Finance,
https://dl.acm.org/doi/abs/10.1145/3383455.3422570, 2020
pages

[27] Thomas Spooner, John Fearnley, Rahul Savani, and Andreas Koukorinis,
Market making via reinforcement learning
https://arxiv.org/abs/1804.04216,
Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, 2018
pages

[28] Mohammad Mani, Steve Phelps, and Simon Parsons,
Applications of Reinforcement Learning in Automated Market-Making
https://nms.kcl.ac.uk/simon.parsons/publications/conferences/gaiw19.pdf,
Proceedings of the GAIW: Games, Agents and Incentives Workshops, Montreal, Canada, 2019
pages

[29] Svitlana Vyetrenko, Shaojie Xu,
Risk-Sensitive Compact Decision Trees for Autonomous Execution in Presence of Simulated Market Response
https://arxiv.org/abs/1906.02312,
Proceedings of the 36th International Conference on Machine Learning, Long Beach, California, 2019
pages

[30] Yuriy Nevmyvaka, Yi Feng, Micheal Kearns,
Reinforcement learning for optimized trade execution
https://dl.acm.org/doi/abs/10.1145/1143844.1143929,
Proceedings of the 23rd international conference on Machine learning, 2006
pages

[31] Auquan, *Evaluating Trading Strategies*
https://medium.com/auquan/evaluating-trading-strategies-fe986062a96b
Medium Blogs, January 2017
pages

[32] Apoorva Singh and Rekhit Pachanekar
Sharpe Ratio: Calculation, Application, Limitations
https://blog.quantinsti.com/sharpe-ratio-applications-algorithmic-trading/#Sortino, QuantInsti Blogs, December 2019
pages

[33] Ctrader FXPro, https://www.fxpro.com/, 2023
pages

[34] TradingFX VPS, https://www.tradingfxvps.com/, 2023
pages

[35] Pytorch, https://pytorch.org/, 2023
pages

[36] Nvidia CUDA, https://developer.nvidia.com/cuda-zone, 2023
pages

[37] OpenAI Gym, https://www.gymlibrary.dev/index.html, 2023
pages

[38] Flask, https://flask.palletsprojects.com/en/2.3.x/, 2023
pages

[39] Paul Billiet, *The Mann-Whitney U-test – Analysis of 2-Between-Group Data with a Quantitative Response Variable*,
https://psych.unl.edu/psycrs/handcomp/hcmann.PDF,
University of Nebraska-Lincoln, 2003
pages

[40] CTrader, *End-User License Agreement*, https://ctrader.com/eula/, 2023
pages

Multi-Agent Reinforcement Learning in a Realistic Limit Order Book Market Simulation

Michaël Karpe*
Jin Fang*
michael.karpe@berkeley.edu
jin_fang@berkeley.edu
University of California, Berkeley
Berkeley, California

Zhongyao Ma†
Chen Wang†
mazy@berkeley.edu
chenwang@berkeley.edu
University of California, Berkeley
Berkeley, California

arXiv:2006.05574v2 [q-fin.TR] 11 Sep 2020

ABSTRACT

Optimal order execution is widely studied by industry practitioners and academic researchers because it determines the profitability of investment decisions and high-level trading strategies, particularly those involving large volumes of orders. However, complex and unknown market dynamics pose significant challenges for the development and validation of optimal execution strategies. In this paper, we propose a model-free approach by training Reinforcement Learning (RL) agents in a realistic market simulation environment with multiple agents. First, we configure a multi-agent historical order book simulation environment for execution tasks built on an Agent-Based Interactive Discrete Event Simulation (ABIDES) [6]. Second, we formulate the problem of optimal execution in an RL setting where an intelligent agent can make order execution and placement decisions based on market microstructure trading signals in High Frequency Trading (HFT). Third, we develop and train an RL execution agent using the Double Deep Q-Learning (DDQL) algorithm in the ABIDES environment. In some scenarios, our RL agent converges towards a Time-Weighted Average Price (TWAP) strategy. Finally, we evaluate the simulation with our RL agent by comparing it with a market replay simulation using real market Limit Order Book (LOB) data.

KEYWORDS

high-frequency trading, limit order book, market simulation, multi-agent reinforcement learning, optimal execution

ACM Reference Format:
Michaël Karpe, Jin Fang, Zhongyao Ma, and Chen Wang. 2020. Multi-Agent Reinforcement Learning in a Realistic Limit Order Book Market Simulation. In *ACM International Conference on AI in Finance (ICAIF '20), October 15–16, 2020, New York, NY, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3383455.3422570

*Both authors contributed equally to this research.
†Both authors contributed equally to this research.

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).
ICAIF '20, October 15–16, 2020, New York, NY, USA
© 2020 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-7584-9/20/10.
https://doi.org/10.1145/3383455.3422570

1 INTRODUCTION

1.1 Agent-based Simulation for Reinforcement Learning in High-Frequency Trading

Simulation techniques lay the foundations for understanding market dynamics and evaluating trading strategies for both financial sector investment institutions and academic researchers. Current simulation methods are based on sound assumptions about the statistical properties of the market environment and the impact of transactions on the prices of financial instruments. Unfortunately, market characteristics are complex and existing simulation methods cannot replicate a realistic historical trading environment. The trading strategies tested by these simulations generally show lower profitability when implemented in real markets. It is therefore necessary to develop interactive agent-based simulations that allow trading strategy activities to interact with historical events in an environment close to reality.

High Frequency Trading (HFT) is a trading method that allows large volumes of trades to be executed in nanoseconds. Execution strategies aim to execute a large volume of orders with minimal adverse market price impact. They are particularly important in HFT to reduce transaction costs. A common practice of execution strategies is to split a large order into several child orders and place them over a predefined period of time. However, developing an optimal execution strategy is difficult given the complexity of both the HFT environment and the interactions between market participants.

The availability of NASDAQ's high-frequency LOB data allows researchers to develop model-free execution strategies based on RL through LOB simulation. These model-free approaches do not make assumptions or model market responses, but rely instead on realistic market simulations to train an RL agent to accumulate experience and generate optimal strategies. However, no existing research has implemented RL agents in realistic simulations, which makes the generated strategies suboptimal and not robust in real markets.

1.2 Related work

The use of RL for developing trading strategies based on large scale experiments was firstly studied by Nevmyvaka et al. in 2006 [13]. The topic has gained popularity in recent years due to the improvements in computing resources, the advancement of algorithms, and the increasing availability of data. HFT makes necessary the use of RL automate and accelerates order placement. Many papers

present such RL approaches, such as temporal-difference RL [15] and risk-sensitive RL [11].

Although RL strategies have proven their effectiveness, they suffer from a lack of explainability. Thus, the need for explaining these strategies in a business context has led to the development of representations of risk-sensitive RL strategies in the form of compact decision trees [19]. Advances in the development of RL agents for trading and order placement then showed the need to learn strategies in an environment close to a real market one. Indeed, traditional RL approaches suffer from two main shortcomings.

First, each financial market agent adapts its strategy to the strategies of other agents, in addition to the market environment. This has led researchers to consider the use of Multi-Agent Reinforcement Learning (MARL) for learning trading and order placement strategies [14] [2]. Second, the market environment simulated in classical RL approaches was too simplistic. The creation of a standardized market simulation environment for artificial intelligence agent research was then undertaken to allow agents to learn in conditions closer to reality, through the creation of ABIDES [6]. Research works on the metrics to be considered to evaluate the agents of RL in this environment was also supported within the framework of LOB simulation [18].

As MARL and ABIDES allow a simulation much closer to real market conditions, additional research was conducted to address the curse of dimensionality, as millions of agents compete in traditional market environments. The use of mean field MARL allows faster learning of strategies by approximating the behavior of each agent by the average behavior of its neighbors [20].

The notion of fairness applied to MARL [9] also brings both efficiency and stability by avoiding situations where agents could act disproportionately in the market, for example by executing large orders. The integration of fairness into MARL [3] has been studied as an evolution of traditional MARL strategies used for example for liquidation strategies [4].

1.3 Contributions

Our main contributions in HFT simulation and RL for optimal execution are the following:

- We set up a multi-agent LOB simulation environment for the training of RL execution agents within ABIDES. Based on existing functionality in ABIDES, such as the simulation kernel and non-intelligent background agents, we develop several RL execution agents and make adjustments to the kernel and framework parameters to suit the change.
- We formulate the problem of optimal execution within an RL framework, consisting of a combination of action spaces, private states, market states and reward functions. To the best of our knowledge, this is the first formulation with optimal execution and optimal placement combined in the action space.
- We develop an RL execution agents using the Double Deep Q-Learning (DDQL) algorithm in the ABIDES environment.
- We train an RL agent in a multi-agent LOB simulation environment. In some situations, our RL agent converges to the TWAP strategy.
- We evaluate the multi-agent simulation with an RL agent trained on real market LOB data. The observed order flow model is consistent with LOB stylized facts.

2 OPTIMAL EXECUTION USING DOUBLE DEEP Q-LEARNING

2.1 Optimal execution formulation

In our work, we allow RL agents not only to choose the order volume to be placed, but also to choose between a market order and one or more limit orders at different levels of the order book. In this section, we describe the states, actions, and rewards of our optimal execution problem formulation.

We define the trading simulation as a T-period problem, which is denoted by the times $T_0 < T_1 < \cdots < T_N$ with $T_0 = 0$. We focus on the time horizon from 10:00am to 3:30pm for each trading day, in order to avoid the most volatile periods of the trading day. Indeed, we observe that it is difficult to train agents to capture complex market behaviors given limited data. The time interval within each period is $\Delta T = 30$ seconds, so that there is a total of 660 periods within the time horizon we define in a trading day, lasting 5 hours (i.e. $T_N = T/\Delta T = 660$). P represents the price, while Q represents the quantity volume at a certain price in the limit order book. Our optimal execution problem is then formulated as follows:

(1) **State** s: the state space includes the information on the LOB at the beginning of each period. For each time period, we use a tuple containing the following characteristics to represent the current state:
   - *time remaining* t: the time remaining after the time period T_k. Since we assume that a trade can only take place at the beginning of each period, this variable also contains the number of remaining trading times. The variable is normalized to be in the $[-1, 1]$ range as follows:
   $$t = 2 \times \frac{T - t}{T} - 1$$
   - *quantity remaining* n: the quantity of remaining inventory at the time period T_k, depending on the initial inventory N, which is also normalized:
   $$n = 2 \times \frac{N - \sum_{i=0}^{t} n_i}{N} - 1$$

   The above state variables are linked to specific execution tasks, called private states. In addition, we also use the following market state variables to capture the market situation at a given point in time:
   - *bid-ask spread* s: the difference between the highest bid price and the lowest ask price, which is intended to provide information on the liquidity of the asset in the market:
   $$s = P_{best\_ask} - P_{best\_bid}$$
   - *volume imbalance* v_{imb}: the difference between the existing order volume on the best bid and best ask price levels. This feature contains information on the current liquidity difference on both sides of the order book, indirectly reflecting the price trend.
   $$v_{\text{imb}} = \frac{Q_{best\_ask} - Q_{best\_bid}}{Q_{best\_ask} + Q_{best\_bid}}$$

- *one-period price return*: the log-return of the stock price over two consecutive days measures the short-term price trend. We intend to allow the RL agent to learn short term return characteristics of the stock price.

$$r_1 = \log\left(\frac{P_t}{P_{t-1}}\right)$$

- *t-period price return*: the log-return of the stock price since the beginning of the simulation measures the deviation between the stock price at time t and the initial price at time 0.

$$r_t = \log\left(\frac{P_t}{P_0}\right)$$

(2) **Action** a**:** the action space defines a possible executed order in a given state, i.e. the possible quantity of remaining inventory affected by the order. In this case, the order can be either a market order or a limit order, either from the bid side or from the ask side. Therefore, the action for each state is a combination of the quantity to be executed with the direction for placement. We use the execution choice to indicate the former and the placement choice to represent the latter.

- *Execution Choices:* At the beginning of each period, the agent decides on an execution quantity N_t derived from the order quantity N_{TWAP} placed using the TWAP strategy. a is taken in $\{0.1, 0.5, 1.0, \ldots, 2.5\}$ and is a scalar the agent chooses from a set of numbers to increase or decrease N_{TWAP}:

$$N_t = a \cdot N_{TWAP}$$

- *Placement Choices:* The agent chooses one of the following order placement methods:
  - choice 0: Market Order
  - choice 1: Limit Order - place 100% on top-level of LOB
  - choice 2: Limit Order - place 50% on each of top 2 levels
  - choice 3: Limit Order - place 33% on each of top 3 levels

(3) **Reward** r**:** the reward intends to reflect the feedback from the environment after agents have taken a given action in a given state. It is usually represented by a reward function consisting of useful information obtained from the state or the environment. In our formulation, the reward function R_t measures the execution price slippage and quantity.

$$R_t = \left(1 - \frac{|P_{fill} - P_{arrival}|}{P_{arrival}}\right) \cdot \lambda \frac{N_t}{N}$$

where λ is a constant for scaling the effect of the quantity component.

2.2 Double Deep Q-Learning algorithm

Aiming to achieve the optimal execution policy, RL enables agents to learn the best action to take through interaction with the environment. The agent follows the strategy that can maximize the expectation of cumulative reward. In Q-Learning, it is represented by the function proposed by Mnih et al. [12]:

$$Q(s, a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \times R(s, a_t, s')\right]$$

However, the above approach would be redundant with larger dimensions of the state space where states cannot be visited in depth. Thus, instead of directly using a matrix as the Q-function, we can learn a feature-based value function $Q(s, a|\theta)$, where θ is the weighting factor that is updated by a stochastic gradient descent.

The Q-value with parametric representation can be estimated by multiple flexible means, in which the Deep Q-Learning (DQL) best fits our problem. In the DQL, the Q-function $Q(s, a|\theta)$ is combined with a Deep Q-Network (DQN), and θ is the network parameters.

The network memory database contains samples with tuples of information recording the current state, action, reward and next state (s, a, r, s'). For each period, we generate samples according to an ε-greedy policy and store them in memory. We replace the samples when the memory database is full, following the First-In-First-Out (FIFO) principle, which means that old samples are removed first.

At each iteration, we batch a given quantity of samples from the memory database, and compute the target Q-value y for each sample, which is defined by Mnih et al. [12] as:

$$y = R(s, a) + \gamma \cdot \max_a Q(s, a|\theta)$$

The network parameter θ is updated by minimizing the loss between the target Q-value and the estimated Q-value calculated from the network, based on current parameters.

However, the DQL algorithm suffers from both instability and overestimation problems, since the neural network both generates the current target Q-value and updates the parameters. A common method to solve this issue is to introduce another neural network with the same structure and calculate the Q-value separately, which is called Double Deep Q-Learning (DDQL) [17].

In DDQL, we distinguish two neural networks, the evaluation network and the target network, in order to generate the appropriate Q-value. The evaluation network selects the best action a^* for each state in the sample, while the target network estimates the target Q-value. We update the evaluation network parameters θ_E every period, and replace the target network parameters θ_T with $\theta_T = \theta_E$ after several iterations.

3 REINFORCEMENT LEARNING IN ABIDES

3.1 ABIDES environment

ABIDES is an Agent-Based Interactive Discrete Event Simulation environment primarily developed for Artificial Intelligence (AI) research in financial market simulations [6].

The first version of ABIDES *(0.1)* was released in April 2019 [6]. The ABIDES development team released a second version *(1.0)* in September 2019, which is supposed to be the first stable release. Finally, the latest version *(1.1)*, released in March 2020, adds many new functionalities, including the implementation of new agents, such as Q-Learning agents, as well as the computation of the realism metrics.

ABIDES aims to replicate a realistic financial market environment by largely replicating the characteristics of real financial markets such as NASDAQ, including *nanosecond time resolution, network latency and agent computation delays and communication solely by means of standardized message protocols* [6]. In addition,

by providing ABIDES with historical LOB data, we are able to reproduce a given period of this history using ABIDES *marketreplay* configuration file.

ABIDES also aims to help researchers in answering questions raised about the understanding of market behavior, such as the influence of delays in sending orders to an exchange, the price impact of placing large orders or the implementation of AI agents into real markets [6].

ABIDES uses a hierarchical structure in order to ease the development of complex agents such as AI agents. Indeed, thanks to *Python object-oriented programming* and *inheritance*, we can, for example, create a new *ComplexAgent* class which inherits from the *Agent* and thus benefits from all functionalities available in the *Agent* class. We can then use *overriding* if we want to change an *Agent* function in order to make it specific for our *ComplexAgent*.

Given that ABIDES is not only for financial market simulations, the base *Agent* class has nothing related to financial markets and is provided only with functions for Discrete Event Simulation. The *FinancialAgent* class inherits from *Agent* and has supplementary functionalities to deal with currencies. On the one hand, the *ExchangeAgent* class inherits from *FinancialAgent* and simulates an financial exchange. On the other hand, the *TradingAgent* also inherits from *FinancialAgent* and is the base class for all trading agents which will communicate with the *ExchangeAgent* during financial market simulations.

Some trading agents – i.e. inheriting from the *TradingAgent* class – are already provided in ABIDES, such as the *MomentumAgent* which places orders depending on a given number of previous observations on the simulated stock price. In the next sections, unless otherwise mentioned, the agents we refer to are all trading agents.

We present in Figure 1 an example of market simulation in ABIDES using real historical data. A *ZeroIntelligenceAgent* places orders on a stock in a market simulation with 100 *ZeroIntelligenceAgent* trading against an *ExchangeAgent*. Each agent is able to place long, short or exit orders, competing with thousands of other agents to maximize their reward.

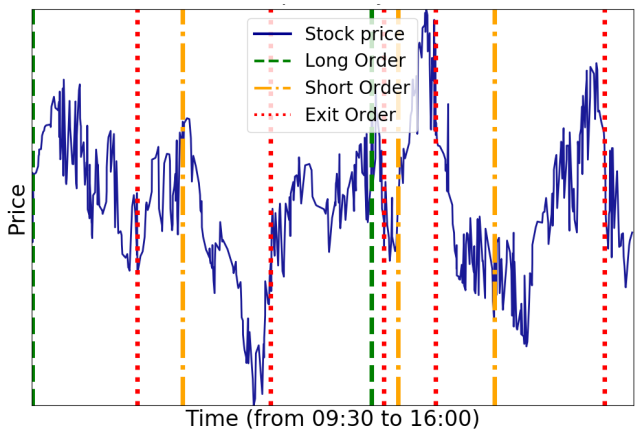


Figure 1: Agent placing orders on simulated stock price

3.2 DDQL implementation in ABIDES

In order to train the DDQL agent in ABIDES during a *marketreplay* simulation, the learning process needs to be integrated with the simulation process. The training process starts by initializing the ABIDES execution simulation kernel and instantiating a *DDQLExecutionAgent* object. The same agent object needs to complete B simulations, B being referred to as the number of training episodes.

Within each training episode, the simulation is divided into N discrete periods. For each period T_i, the agent chooses an action a_{T_i} for the current period according to the ε-greedy policy in order to achieve a balance of exploration and exploitation. Then, an order schedule is generated, based on the quantity and placement strategy defined in the chosen action. The current-period order could be broken into small orders and placed on different levels of the LOB. Then, the current experience $\left(s_{T_i}, a_{T_i}, s_{T_{i+1}}, r_{T_i}\right)$ is stored in the replay buffer $\mathcal{D}$.

The replay buffer removes the oldest experience when its size reaches to the maximum capacity specified. This intends to use relatively recent experiences to train the agent. As long as the size of the replay buffer $\mathcal{D}$ reaches a minimum training size, a random minibatch $\left(s_{(j)}, x_{(j)}, r_{(j)}, s^{s,x}_{(j)}\right)$ is sampled from $\mathcal{D}$ for training the evaluation network.

The target network is updated after training the evaluation network 5 times. This training frequency is chosen with a small grid search on multiple training frequencies, balancing computing burden, model fitting performance, and experience accumulation. The final step within time period T_i is to update the state $s_{T_{i+1}}$ and compute the reward r_{T_i} for the current period. The entire process is summarized in the algorithm below.

Algorithm 1 Training of DDQL for optimal execution in ABIDES.

for training episode $b \in B$ **do**
  for $i \leftarrow 0$ to $N-1$ **do**
    With probability ε select random action a_{T_i}
    Otherwise select optimal action $a_{T_{i'}}$ based on target_net
    Schedule orders o_i according to a_i and submit o_i
    Store experience $\left(s_{T_i}, a_{T_i}, s_{T_{i+1}}, r_{T_i}\right)$ in replay buffer $\mathcal{D}$
    if $length(\mathcal{D}) > max\_experience$ **then**
      Remove oldest experience
    if $length(\mathcal{D}) \geq min\_experience$ AND $i \mod 5 == 0$ **then**
      Sample random minibatch from $\mathcal{D}$
      Train eval_net and update target_net
    if orders o_i accepted or executed **then**
      Observe environment and update $s_{T_{i+1}}$
      Compute and update r_{T_i}

To train a DDQL agent, we implement our neural network based on a Multi Layer Perceptron (MLP). We stack multiple dense layers together as illustrated in Figure 2, and we set the activation function to be ReLU to introduce non-linearity. Dropout is added to avoid overfitting. The optimization algorithm chosen for backpropagation is the *Root Mean Square back-propagation* (RMSprop) proposed by Tieleman et al. [16] with a learning rate of 0.01. The loss function

we choose is the *mean squared error* (MSE). The size of the output layer size is the number of actions to choose from.

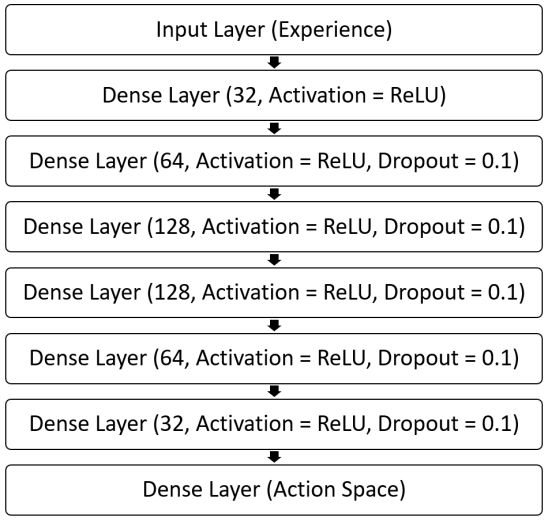


Figure 2: Multi-Layer Perceptron (MLP) architecture

4 EXPERIMENTS AND RESULTS

In this section, we describe our experiment for training a DDQL agent in a multi-agent environment and observe the behavior of the agent during testing.

4.1 Data for experiments

The data we use for the experiments are NASDAQ data converted to LOBSTER [8] format to fit the simulation environment. We extract the order flow for 5 stocks (CSCO, IBM, INTC, MSFT and YHOO) from January 13 to February 6, 2003. We train the model over 9 days and test it over the following 9 days. The training data is concatenated into a single sequence, and the training process is continuous for consecutive days while the model parameters are stored in intermediate files.

4.2 Multi-agent configuration

The multi-agent environment that we have set up for the training of DDQL agents at ABIDES consists of an *ExchangeAgent*, a *MarketreplayAgent*, six *MomentumAgents*, a *TWAPExecutionAgent* and our *DDQLExecutionAgent*.

- *ExchangeAgent* acts as a centralized exchange that keeps the order book and matches orders on the bid and ask sides.
- *MarketreplayAgent* accurately replays all market and limit orders recorded in the historical LOB data.
- *MomentumAgent* compares the last 20 mid-price observations with the last 50 mid-price observations and places a buy limit order if the 20 mid-price average is higher than the 50 mid-price average, or a sell limit order if it is.
- *TWAPAgent* adopts the TWAP strategy. This strategy minimizes the price impact by dividing a large order equally into several smaller orders. Its execution price is the average price of the recent k time periods. The agent's optimal trading rate is calculated by dividing the total size of the order by the total execution time, which means that the trading quantity is constant. When the stock price follows a Brownian motion and the price impact is assumed to be constant, this strategy is optimal [7]. In RL, if there is no penalty for any running inventory, but a significant penalty for the ending inventory, the TWAP strategy is also optimal.

4.3 Result of our experiments

We observe that our RL agent converges to the TWAP strategy after 9 consecutive days of training regardless of the stock chosen. The agent places a top-level limit order or market order. However, the execution quantity chosen by the agent throughout the test period changes with the stock. This result may be explained by the fact that our RL agent places an order every 30 seconds, and thus is not able to capture the trading signals existing during shorter periods of time.

5 REALISM OF OUR LOB SIMULATION

Numerous research papers studied the behaviour of the LOB. A recent research paper presents a review of these LOB characteristics which can be referred to as *stylized facts*, as reminded in Vyetrenko et al. [18] In this section, we compare our simulation with real markets based on these *realism metrics* in order to assess whether our market simulation, mainly built on ABIDES, is realistic.

We can mainly distinguish two sets of metrics for the analysis of the LOB behavior. The first set includes metrics related to asset return distributions and the second set includes metrics related to volume and order flow distributions [18].

Asset returns metrics generally relate to price return or percentage change. For the LOB, it includes the mid-price trend, which is the average of the best bid price and the best ask price. Volumes and order flow metrics, for their part, relate to the behavior of incoming order flows, including new buy orders, new sell orders, order modifications or order cancellations. Three main stylized facts related to order flows are as follows:

- **Order volume in a fixed time interval:** Order volume in a fixed interval or time window likely follows a positively skewed log-normal distribution or gamma distribution [1].
- **Order interarrival time:** The time interval of two consecutive limit orders likely follows an exponential distribution [10] or a Weibull distribution [1].
- **Intraday volume patterns:** Limit order volume within a given time interval for each trading day can be approximated by a U-shaped polynomial curve, where the volume is higher at the start and the end of the trading day [5].

These realism metrics are implemented in ABIDES. We compute the stylized facts implemented in ABIDES on a *marketreplay* simulation with and without the presence of our *DDQLExecutionAgent*. For all of the stylized facts, we observe that adding a single new agent to a simulation does not significantly alter the result of the computation. This means that evaluating the realism of our simulation with a single *DDQLExecutionAgent* is equivalent to evaluating the realism of the LOB data provided as an input to the simulation.

On our NASDAQ LOB 2003 data, we always observe the two first order flow stylized facts mentioned above, but not the intraday volume patterns. As an example of successfully observed stylized fact, Figure 3 illustrates the stylized fact about order volume in a fixed time interval for the IBM stock on January 13, 2003. We verify that order volume in a fixed time interval follows a gamma distribution. It shows that the most likely order volume value in the 60-second window is very low, while the volume average can be high.

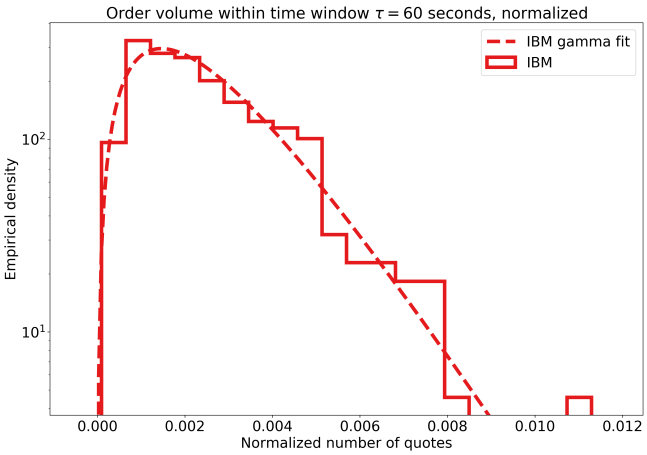


Figure 3: Order volume in fixed time interval for IBM stock on January 13, 2003

6 CONCLUSION AND FUTURE WORK

6.1 Conclusion

In this paper, we built our *DDQLExecutionAgent* in ABIDES by implementing our own optimal execution problem formulation through RL in a financial market simulation, and set up a multi-agent simulation environment accordingly. In addition, we conducted experiments to train our *DDQLExecutionAgent* in the ABIDES environment and compared the agent strategy with the TWAP strategy to which our agent converges. Finally, we evaluated the realism of our multi-agent simulations by comparing LOB stylized facts on simulations using our *DDQLExecutionAgent* with the ones of a *marketreplay* simulation using real LOB data.

6.2 Future work

Due to limited computing resources and lack of data, the experiments we have been able to conduct are limited. Our current model can be improved in many ways. First, instead of training agents on steady periods (10:00am to 3:30pm) only, we can train agents on entire trading days. The agent period of time that we set can be refined to a time interval closer to the nanosecond, as in real HFT. Second, the action space can be expanded to include more types of execution actions and the reward function can be enhanced to include more information and feedback from both the market and other agents. With respect to the state's feature, we choose log price to explicitly capture the mean reversion in financial market which may easily overestimate the effect. Therefore, a more appropriate indicator should be chosen to modify this situation.

Regarding the RL algorithm, we can try several advanced methods to implement an updated approach on our *DDQLExecutionAgent*. Directions include the use of prioritized experience replay to increase the frequency of batching important transitions from memory, or the combination of bootstrapping with DDQL to improve exploration efficiency. Other methods such as increasing the complexity of the network architecture will only be useful if implemented in a more complex environment. Several LSTM layers can be added to take advantage of the agent's past experience and improve the performance when providing a larger training data set or applying a longer training time period.

So far, we focused on a relatively monotonous set of multiple agents, which is not able to fully capture the influence of the interaction between the agents. To remedy this situation, more and different types of agents can be added to the configuration to study collaboration and competition among agents in further detail. Moreover, after introducing a more complex combination of agents in the ABIDES environment, we can try to perform the financial market simulation on the basis of this configuration, which should be much more realistic than the existing one.

The approach to evaluation is also an aspect that can be further expanded. Our experience does not currently allow us to clearly distinguish the difference between our agents and the benchmark. In order to assess the model more accurately, we can further improve our evaluation methods to examine both the parameters of RL and financial performance. For example, by conducting a simulation of the financial market in the ABIDES environment, we can use the realism metrics we have designed to evaluate our agents.

In our experiments, our *DDQLExecutionAgent* learns how to perform a TWAP strategy because its trading frequency is not high enough. However, this work shows the potential of MARL for developing optimal trading strategies in real financial markets, by implementing agents with an higher trading frequency in realistic market simulations.

ACKNOWLEDGMENTS

We would like to thank Prof. Xin Guo and Yusuke Kikuchi for their helpful comments during the realization of this work. We also want to thank Svitlana Vyetrenko, Ph.D. for her answers to our questions about the work she contributed to and cited in the References section.

REFERENCES

[1] Frédéric Abergel, Marouane Anane, Anirban Chakraborti, Aymen Jedidi, and Ioane Muni Toke. 2016. *Limit order books.* Cambridge University Press.
[2] Tucker Hybinette Balch, Mahmoud Mahfouz, Joshua Lockhart, Maria Hybinette, and David Byrd. 2019. How to Evaluate Trading Strategies: Single Agent Market Replay or Multiple Agent Interactive Simulation? *arXiv preprint arXiv:1906.12010* (2019).
[3] Wenhang Bao. 2019. Fairness in Multi-agent Reinforcement Learning for Stock Trading. *arXiv preprint arXiv:2001.00918* (2019).
[4] Wenhang Bao and Xiao-yang Liu. 2019. Multi-agent deep reinforcement learning for liquidation strategy analysis. *arXiv preprint arXiv:1906.11046* (2019).
[5] Jean-Philippe Bouchaud, Julius Bonart, Jonathan Donier, and Martin Gould. 2018. *Trades, quotes and prices: financial markets under the microscope.* Cambridge University Press.
[6] David Byrd, Maria Hybinette, and Tucker Hybinette Balch. 2019. Abides: Towards high-fidelity market simulation for ai research. *arXiv preprint arXiv:1904.12066* (2019).

[7] Kevin Dabérius, Elvin Granat, and Patrik Karlsson. 2019. Deep Execution-Value and Policy Based Reinforcement Learning for Trading and Beating Market Benchmarks. *Available at SSRN 3374766* (2019).

[8] Ruihong Huang and Tomas Polak. 2011. LOBSTER: Limit order book reconstruction system. *Available at SSRN 1977207* (2011).

[9] Jiechuan Jiang and Zongqing Lu. 2019. Learning Fairness in Multi-Agent Systems. In *Advances in Neural Information Processing Systems*. 13854–13865.

[10] Junyi Li, Xintong Wang, Yaoyang Lin, Arunesh Sinha, and Michael P Wellman. 2018. Generating Realistic Stock Market Order Streams. (2018).

[11] Mohammad Mani, Steve Phelps, and Simon Parsons. 2019. Applications of Reinforcement Learning in Automated Market-Making. (2019).

[12] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[13] Yuriy Nevmyvaka, Yi Feng, and Michael Kearns. 2006. Reinforcement learning for optimized trade execution. In *Proceedings of the 23rd international conference on Machine learning*. 673–680.

[14] Yagna Patel. 2018. Optimizing Market Making using Multi-Agent Reinforcement Learning. *arXiv preprint arXiv:1812.10252* (2018).

[15] Thomas Spooner, John Fearnley, Rahul Savani, and Andreas Koukorinis. 2018. Market making via reinforcement learning. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 434–442.

[16] Tijmen Tieleman and Geoffrey Hinton. 2012. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning* 4, 2 (2012), 26–31.

[17] Hado Van Hasselt, Arthur Guez, and David Silver. 2016. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI conference on artificial intelligence*.

[18] Svitlana Vyetrenko, David Byrd, Nick Petosa, Mahmoud Mahfouz, Danial Dervovic, Manuela Veloso, and Tucker Hybinette Balch. 2019. Get Real: Realism Metrics for Robust Limit Order Book Market Simulations. *arXiv preprint arXiv:1912.04941* (2019).

[19] Svitlana Vyetrenko and Shaojie Xu. 2019. Risk-Sensitive Compact Decision Trees for Autonomous Execution in Presence of Simulated Market Response. *arXiv preprint arXiv:1906.02312* (2019).

[20] Yaodong Yang, Rui Luo, Minne Li, Ming Zhou, Weinan Zhang, and Jun Wang. 2018. Mean field multi-agent reinforcement learning. *arXiv preprint arXiv:1802.05438* (2018).

arXiv:2307.04863v2 [q-fin.TR] 27 Sep 2024

Interpretable ML for High-Frequency Execution

Timothee FABRE*†*and Vincent RAGEL*†

September 30, 2024

Abstract

Order placement tactics play a crucial role in high-frequency trading algorithms and their design is based on understanding the dynamics of the order book. Using high quality high-frequency data and a set of microstructural features, we exhibit strong state dependence properties of the fill probability function. We train a neural network to infer the fill probability function for a fixed horizon. Since we aim at providing a high-frequency execution framework, we use a simple architecture. A weighting method is applied to the loss function such that the model learns from censored data. By comparing numerical results obtained on both digital asset centralized exchanges (CEXs) and stock markets, we are able to analyze dissimilarities between feature importances of the fill probability of small tick crypto pairs and Euronext equities. The practical use of this model is illustrated with a fixed time horizon execution problem in which both the decision to post a limit order or to immediately execute and the optimal distance of placement are characterized. We discuss the importance of accurately estimating the clean-up cost that occurs in the case of a non-execution and we show it can be well approximated by a smooth function of market features. We finally assess the performance of our model with a backtesting approach that avoids the insertion of hypothetical orders and makes possible to test the order placement algorithm with orders that realistically impact the price formation process.

Keywords – Optimal Execution; Fill Probability; Survival Analysis; Limit Order Book; High Frequency

Contents

*timothee.fabre@centralesupelec.fr
†vincent.ragel@centralesupelec.fr

1 Introduction

Estimating one's execution probability when providing liquidity is a must either for market making or for optimal execution. Recent promising methods leverage complex deep learning architectures applied to raw data or to synthetic data when data is missing or censored. Yet, using complex features and simple neural architectures is both faster and more interpretable.

In continuous double-auction markets, one can choose to add liquidity by posting a limit order, which carries the risk of potentially not being executed if the market price does not reach the limit price, or to take liquidity by sending a marketable order, which is more expensive due to the crossing of the spread. Depending on the agent's objective and utility function, a trading algorithm may favor one order type over the other and adjust the price of the limit order according to some predefined rules. For example, a market making strategy manages the inventory risk and avoids adverse selection by adjusting the posting distances around a fair price. An execution algorithm seeks for a discounted execution price on one side of the book while aiming at a target quantity to execute under a specific time horizon that can be arbitrarily small, *i.e.* of the order of milliseconds to seconds. Although these strategies are of different nature, they share a central source of uncertainty that is the randomness of execution.

Ultimately, deciding between sending a limit or a market order requires the knowledge of the execution probability of a limit order. The latter depends on market variables such as the bid-ask spread, the volatility, the order flow regime, etc. The order flow itself is highly sensitive to many market variables and they may be clustered in two classes of features: *snapshot variables* which compose the Markovian part of the model (bid-ask spread for example) and *differential variables* which compose the non-Markovian part of the model (traded volume variation computed over a period or realized volatility for example). A fill probability model should find its predictive power in both classes of variables in order to capture the main explanatory features of the order flow dynamics.

When opting for the placement of a limit order in the book, the non-execution risk combined with price volatility may cause additional costs —the clean-up cost— if the target quantity to trade is not reached at the end of the time horizon. Assuming that the target quantity must be traded at all costs, a marketable order sent at the end of the period will potentially trade at a worse price than if it had been sent at the beginning due to adverse price movements. The estimation of such a price move in the case of non-execution is capital since it draws the decision boundary of the order placement tactic, and its sensitivity to market variables leads to a complex interaction with the fill probability function in the decision-making process. As an example, one easily intuits that both fill probability and clean-up cost functions increase with the realized volatility which will force the agent to make a trade-off between market risk and execution likelihood. Therefore, to decide between a limit order and a market order, one must properly evaluate both the execution probability and the clean-up cost.

1.1 Literature review

The fill probability function estimation is generally carried out using two main classes of methods. The first class encompasses survival analysis tools and is applied to financial data in Cho and Nelling (2000); Lo et al. (2002), that carry out empirical analysis of the role played by multiple key market variables. The second class of estimation procedure encompasses the insertion of hypothetical limit orders and the computation of their first passage time, either using transaction data or the crossing of a reference price. First passage time distributions of non-Gaussian dynamics are widely used in the literature and their empirical scaling properties are carefully studied in Perelló et al. (2011). The tail exponents of the first passage time and empirical time-to-fill and time-to-cancel are analyzed in Eisler et al. (2012) and the authors suggest that the fatter tails observed for the first passage times are explained by cancellations. They also provide a simple model that succeeds in capturing the above stylized facts. A state dependent model of the fill probability function was proposed in Maglaras et al. (2022) for which a recurrent neural network was used to compute the fill probability of synthetic limit orders as a function of market features. In a more recent work (Arroyo et al., 2024), the authors develop a deep neural network structure for the estimation of the full survival function. They use the right-censored likelihood as a loss function for training and proper scoring rules for model performance assessment. Although they apply the model to market data, the only interpretable features they use are the spread, the volatility and the best queue imbalance. They provide a feature importance analysis, but the small number of predictions used for the computation of the Shapley values as well as the use of raw volumes and prices as inputs leads to a lack of interpretability in the results.

In Maglaras et al. (2022), the authors point out the fact that using real limit orders for fill probability computation brings a selection bias to the analysis, which finds its nature in the heterogeneity of information contained in the order flow. Since limit orders are posted by both informed and uninformed traders with various strategies and time scales, the analysis of such orders cannot be used for the purpose of fill probability computation of an uninformed agent with a specific horizon. The problem is that in practice, the insertion of a limit order impacts the order book and consequently the price process itself, *e.g.* see Weber and Rosenow (2005); Hautsch and Huang (2012); Bacry et al. (2016); Said et al. (2017); Brogaard et al. (2019). Last but not least, the posted size affects market depth and order flow imbalances and thus the execution probability itself. This is even more true in high frequency trading, and the classical first passage time method fails at capturing these stylized facts as the main hypothesis is the absence of impact of posted limit orders. Following the findings of Lo et al. (2002) and the above reasoning, we decide to favor market impact adjusted fill probabilities, *i.e.* real order flow fill probabilities over the removal of selection bias.

Optimal limit order placement tactics are studied in various frameworks (Wald and Horrigan, 2005; Avellaneda and Stoikov, 2008; Laruelle et al., 2013; Bayraktar and Ludkovski, 2014; Markov, 2014; Cartea and Jaimungal, 2015; Cont and Kukanov, 2017). The LOB dynamics are often characterized with an execution intensity that is a decreasing function of the distance. An univariate representation of the execution probability is certainly unrealistic and restrictive, but it still provides enough information to develop order placement algorithms. In Lehalle and Mounjid (2017), the authors build a stochastic control framework and study the impact of adverse selection risk and latency on optimal order placement algorithms. Work has also been done to tackle the queue position valuation problem, see Moallemi and Yuan (2016); Donnelly and Gan (2018).

In the field of machine learning, recent works have proposed state-of-the-art architectures to tackle survival analysis in high dimension. Lee et al. (2018) have proposed a deep neural network for competing risks frameworks and Katzman et al. (2018) have introduced a more sophisticated approach that integrates the classical Cox model framework with the power of deep learning. More recently Wang and Sun (2022) have proposed a model using transformers designed to handle competing events. We decide to opt for simple feed-forward network architecture instead, as we aim at proposing execution frameworks that are suitable to high-frequency trading. Even though complex architectures can perform better, they are generally not adapted to live HF trading environments where an extra microsecond of tick-to-trade latency due to model computations can lead to significant losses.

1.2 Contribution and organization of the paper

Instead of using sophisticated deep learning models in order to extract relevant features but may be difficult to train, our approach uses a combination of handcrafted features and simple feed-forward networks to capture key dependencies while being easily integrated in a decision-making process for execution. This choice simplifies the model structure, leading to improved transparency and better reproducibility of the results.

Our contributions are as follows:

- We define three new microstructure features for fill probability prediction: the limit order flow imbalance, the aggressiveness index and the priority volume. Using survival analysis methods, we provide empirical evidence of a smooth dependence of the fill probability with respect to these variables.
- We apply the inverse-probability-of-censoring weighting method (IPCW) to the training of a neural network, enabling it to learn from censored data.
- A detailed feature importance analysis is provided, demonstrating key differences between small tick cryptocurrencies and Euronext equities regarding the predictive power of interpretable variables. Moreover, to the best of our knowledge, our work is the first to analyze the feature importance of the fill probability depending on the placement of the order, *i.e.* if the order is placed inside the spread, at the current best queue, or deep in the book.
- We set an optimal execution problem where the agent needs to choose between posting a limit order and sending a marketable order. We propose a new backtest methodology that, by essence, takes into account the market impact of limit order insertion for better performance assessment.

The structure of this work is as follows: Section 3 presents the high-frequency data sets that will be used in the numerical experiments. Section 4 discusses two survival analysis methods to compute fill probability on level 3 data and provides empirical evidence of smooth non-linearity of the fill probability as a function of market features. Once these ingredients are gathered, we present our fill probability model in Section 5 and finally study an order placement algorithm that incorporates both the fill probability function and a clean up cost model. Results about the optimal placement policy and an execution-specific backtesting approach are displayed before discussing the importance of latency and how it can be integrated in the cost function.

2 Data

Our work uses two high-frequency data sets with Level-3 granularity (full order details) for digital assets and a mature equity market.

The digital asset data is provided by SUN ZU Lab's proprietary feed handlers. There are few centralized exchanges who provide a market by order data API amongst which Coinbase, Bitstamp and Bitfinex are the most popular. We chose to use Coinbase data for two reasons. First, the tick size remains very small during the covered period of time which is not the case for Bitstamp for example where tick sizes have been enlarged in 2022. Secondly, Coinbase provides a timestamp with microsecond precision and a sequence number with each message allowing us to be confident about the order book reconstruction process. We have 1 month of data at our disposal, from 2022-11-05 to 2022-12-05 on BTC-USD and ETH-USD. We removed several days (2022-11-07 to 2022-11-10) from our analysis because of the extreme volatility that was observed during this period. The equity data comes from BEDOFIH (Base Européenne de Données Financières à Haute-fréquence), built by the European Financial Data Institute (EUROFIDAI). This comprehensive data base offers highly detailed order data for all stocks traded on Euronext Paris between 2013 and 2017. To conduct our analysis, we will concentrate on the most recent year available, from January 2017 to December 2017, for two liquid French stocks, BNPP and LVMH. These are Level 3 data feeds which contain the full sequence of order-based events. This makes it possible to trace the life of each order and identify when it was completed or canceled. However, it requires to reconstruct the order book from individual order events. Table 1 summarizes several descriptive statistics of the data. The spread is clearly different between small tick assets (BTC-USD, ETH-USD) and large tick assets (BNPP, LVMH). The tick size is 0.01 USD for BTC-USD and ETH-USD pairs representing roughly 10^{-6} of BTC-USD and 10^{-5} of ETH-USD. The tick size depends on the price value for both BNPP and LVMH with respect to MIFID rules. In Huang et al. (2016), Laruelle et al. (2019), the differentiation is made for an average spread limit of 1.6 ticks but we still classify BNPP as a large tick asset with respect to digital assets.

We display some descriptive statistics of the lifetime of orders in Table 2, expressed in number of seconds. We observe that most of the orders of the cryptocurrency pairs have significantly smaller lifetimes than those of the equities. The discrepancy between the median and the average indicates the presence of fat tailed distributions.

3 Non-parametric analysis of the execution and cancellation risks

The role of this section is to introduce three new microstructural features and show that they strongly influence both executions and cancellations. A non-parametric estimation is carried out using a competing risks framework that we first briefly review.

3.1 Cancellation as a competing risk

In the rest of the paper, we place ourselves in a filtered probability space $(\Omega, \mathcal{F}, \mathbb{F} := (\mathcal{F}_t)_t, \mathbb{P})$. Let the state of a pending limit order be modeled by the $\mathbb{F}$-adapted process $(X_t)_{t\geq 0}$ with values in a discrete state space $\mathbb{S}$ and such that $X_0 = 0$ *a.s.*. The order is said to be alive at time t if $X_t = 0$ and dead if $X_t \neq 0$. Hence, the lifetime of this order is fully characterized by the random variable $L := \inf\{t > 0, X_t \neq 0\}$. Naturally, all the states (except for 0) are absorbing since they all signify the death of the underlying order. We now discuss the specification of the state space $\mathbb{S}$ in the continuous trading paradigm. We will also denote, for $t \geq 0$, $F(t) := \mathbb{P}(L \leq t)$ and $S(t) := 1 - F(t)$ respectively the cumulative distribution

Table 1: *Descriptive statistics* — Spread, trade size and daily volume statistics

		Spread[a]	**Trade size**[b]	**Daily volume**[c]
BTC-USD	5%	84.31	1.01	197,146,101
	Median	163.40	127.86	470,617,696
	95%	394.25	8,619.66	1,635,066,155
	Mean	192.51	1,725.54	612,757,871
ETH-USD	5%	8.31	0.84	221,517,905
	Median	15.61	371.70	431,407,671
	95%	37.34	7,915.72	1,459,323,150
	Mean	18.07	1,836.98	563,289,467
BNPP	5%	1.00	20	496,135
	Median	2.00	140	194,8473
	95%	3.00	386	3,716,829
	Mean	1.70	180	2,092,348
LVMH	5%	1.00	7	99,870
	Median	1.00	49	319,678
	95%	2.00	147	573,315
	Mean	1.24	61	324,954

[a] Bid-ask spread is expressed in number of ticks.
[b] Trade size is expressed USD for crypto pairs and in number of stocks for the equities. The metrics are computed on the volume of the recorded market orders and serves as a reference for the average trade size (ATS).
[c] Daily volume is expressed in USD for crypto pairs and in number of stocks for the equities.

Table 2: *Descriptive statistics* — Time-to-event statistics, expressed in seconds

		Lifetime	**Time to fill**	**Time to cancel**
BTC-USD	5%	0.003	0.0005	0.003
	Median	0.063	0.125	0.059
	95%	8.499	5.697	7.107
	Mean	152.931	53.672	4.176
ETH-USD	5%	0.003	0.0006	0.003
	Median	0.069	0.205	0.065
	95%	9.035	11.274	7.839
	Mean	117.453	65.914	4.347
BNPP	5%	0.0004	0.0007	0.0004
	Median	1.296	4.069	1.193
	95%	100.08	106.56	96.31
	Mean	48.19	70.73	32.68
LVMH	5%	0.0002	0.0009	0.0002
	Median	4.173	8.375	3.983
	95%	292.92	231.69	281.28
	Mean	85.13	84.91	70.16

function and the survival function (or complementary cumulative distribution function) of the order lifetime L.

This limit order is seen as a birth-death entity for which death —removal from the order book— can be triggered either by its full execution or by its cancellation. It is noteworthy that the case of partial execution is not considered here and should be tackled differently since it is not an absorbing state, the order remaining alive in the book. When the death of the order is not observed, we say that the order is right-censored. Censoring is present in both financial markets and crypto CEXs data sets. An example of a censored order is when it is neither cancelled nor fully executed at the end of the observation period. From the practitioner's point of view, the one of the main causes of censoring in CEXs is feed handler disconnections that may happen depending on the venue (this also happens on traditional financial markets but to a lesser extent). In the case of a disconnection, data feed is stopped for a random time that ranges from milliseconds to seconds or even hours for API shutdowns. The censoring issue in the equity dataset occurs at the close of the trading day and is therefore much smaller.

As outlined in Eisler et al. (2009), the presence of cancellations plays a significant role in the difference observed between empirical first passage time (FPT) and time to fill (TTF) distributions, leading to fatter tails for the FPT. Since we want to analyze both execution and cancellation probabilities, we treat cancellation as a competing risk with respect to execution rather than as censoring since once the order is cancelled, its future execution cannot happen anymore. In fact, when censoring occurs, the event of interest may still happen afterward, but it goes unobserved.

Based on this, we specify the state space as $\mathbb{S} := \{0, 1, 2\}$ and define $\mathbb{S}^\dagger := \{1, 2\}$ the death state space, where 1 indicates a fully executed order and 2 indicates a cancelled order. We define the cause-specific hazard rate for $t \geq 0$, $i \in \mathbb{S}^\dagger$

$$\lambda_i(t) := \lim_{\Delta t \to 0^+} \frac{\mathbb{P}\big(L \in [t, t + \Delta t[, X_L = i | L \geq t\big)}{\Delta t}. \tag{1}$$

We define the cumulative incidence function (CIF), for $t \geq 0$, $i \in \mathbb{S}^\dagger$

$$F_i(t) := \mathbb{P}\left(L \leq t, X_L = i\right) \tag{2}$$

$$= \int_0^t \mathbb{P}\left(L > s\right) \lambda_i(s) \mathrm{d}s, \tag{3}$$

where the last equality is established with elementary calculus using Equation (1).

3.2 Non-parametric estimation

Assume that we either observe the lifetime or the censoring of N limit orders which are exposed to the following mutually exclusive causes of death: execution and cancellation.

Previous notations are naturally indexed in the following manner: we denote the lifetime of the nth order by L_n and the distribution function of L_n is denoted by F; we assume that the order can be subject to independent right censoring modeled by a random variable C_n and the censoring variable is assumed independent from L_n. Thus, what we effectively observe is the realization of the random variables $T_n := \min(L_n, C_n)$ and $\mathbb{1}_{\{L_n \leq C_n\}}$ and t_n denotes our observation of the censored lifetime T_n.

Let $\left(t_{(k)}\right)_{1 \leq k \leq K}$ be the ordered sequence of observed and censored lifetimes such that for $1 \leq n \leq N$, $t_n \in \{t_{(k)}, 1 \leq k \leq K\}$. In the seminal paper Kaplan and Meier (1958), the Kaplan-Meier estimator was introduced as a non-parametric estimator of the survival function in the presence of censored data and is written as

$$\forall\, t \geq 0, \quad \widehat{S}(t) = \prod_{k, t_{(k)} < t} \left(1 - \frac{d_k}{n_k}\right), \tag{4}$$

where d_k is the number of deaths at time $t_{(k)}$ and n_k is the number of pending orders at time $t_{(k)}^-$. Thus, after each $t_{(k)}$, the number of pending orders becomes $n_{k+1} = n_k - (c_k + d_k)$ where c_k is the number of right censored observations occurring at $t_{(k)}$.

A non-parametric estimator of the cumulative incidence function was proposed in Aalen (1976) and Aalen and Johansen (1978). The Aalen-Johansen estimator is defined, for $t > 0$, and $i \in \mathbb{S}^\dagger$ as

$$\widehat{F}_i(t) = \sum_{k,t_{(k)}<t} \widehat{S}\left(t_{(k-1)}\right) \frac{d_k^i}{n_k} \tag{5}$$

using the convention $t_{(0)} := 0$ and denoting by d_k^i the number of deaths from cause i observed at $t_{(k)}$. Naturally, the Kaplan-Meier curve $\widehat{S}(.)$ is computed without distinguishing between the causes of death, such that for $1 \leq k \leq K$, $d_k = d_k^1 + d_k^2$. The Aalen-Johansen estimator is built by summing the product of the Kaplan-Meier survival function and increments of the Nelson-Aalen cumulative hazard rate estimator, see Aalen (1978), Nelson (1969) and Nelson (1972).

Counting process theory provides mathematical expressions of confidence intervals for the Aalen-Johansen estimator. The procedure used here is based on the Gray estimator Pintilie (2006) of the variance of the CIF, written for $t > 0$, $i \in \mathbb{S}^\dagger$,

$$\begin{aligned}\widehat{\mathrm{Var}}\left(\widehat{F}_i(t)\right) = &\sum_{k,t_{(k)}<t} \frac{\left(\widehat{F}_i(t) - \widehat{F}_i\left(t_{(k)}\right)\right)^2 d_k}{(n_k-1)(n_k-d_k)} + \sum_{k,t_{(k)}<t} \frac{\widehat{S}\left(t_{(k-1)}\right)^2 d_k^i m_k^i}{(n_k-1)n_k} \\ &- 2\sum_{k,t_{(k)}<t} \frac{\left(\widehat{F}_i(t) - \widehat{F}_i\left(t_{(k)}\right)\right)\widehat{S}\left(t_{(k-1)}\right) d_k^i m_k^i}{(n_k-d_k)(n_k-1)},\end{aligned} \tag{6}$$

where we used the notation $m_k^i := \dfrac{n_k - d_k^i}{n_k}$.

This estimator is known to over-estimate the true variance (Braun and Yuan, 2007) slightly. As suggested by Kalbfleisch and Prentice (2011), we use the log-log method to compute the confidence interval. This methodology restricts the boundaries of the confidence interval to $[0, 1]$ as this constraint may break when linear confidence intervals are computed. Let α be the confidence level and z_x the Gaussian x-percentile. If we define, for $t > 0$ and $i \in \mathbb{S}^\dagger$

$$C_\alpha^i(t) := z_{\frac{\alpha}{2}} \frac{\sqrt{\widehat{\mathrm{Var}}\left(\widehat{F}_i(t)\right)}}{\widehat{F}_i(t)\log\left(\widehat{F}_i(t)\right)}, \tag{7}$$

then an α-confidence interval $\mathrm{CI}_\alpha(\widehat{F}_i(t))$ is obtained using

$$\mathrm{CI}_\alpha\left(\widehat{F}_i(t)\right) := \left[\widehat{F}_i(t)^{e^{-C_\alpha^i(t)}}, \widehat{F}_i(t)^{e^{C_\alpha^i(t)}}\right]. \tag{8}$$

3.3 Empirical analysis

In the following experiments, we analyze both passive limit order flow, i.e. limit orders that do not affect the bid-ask spread when posted, and aggressive limit order flow, *i.e.* limit orders which form a new best queue when posted. In the latter case, marketable limit orders that are partially executed are not studied and are left for further investigation; note that they could be treated as a special case of aggressive limit orders since they change both best prices. We decide to discard orders that were posted too far in the book and set the threshold at 10% on either side of the mid price for cryptocurrencies. As regards equities, we focus on the first 10 limits of the limit order book.

Our goal is to investigate the influence of microstructural features on the fill probability. To this effect, the order fill time horizon must be small, but sufficiently large to be able to characterize the execution risk. Choosing a too short time horizon for the fill probability would lead to extremely imbalanced classes, resulting in high (relative) estimation error. The time horizon depends on the asset considered and especially on the frequency of trades. We observed that a 1 second time horizon for the crypto pairs and a 10 seconds time horizon for the equities gave satisfactory and comparable results. Given these time horizons, the percentage of observed executions amongst posted limit orders is 2% for the crypto pairs and 4% for the equities.

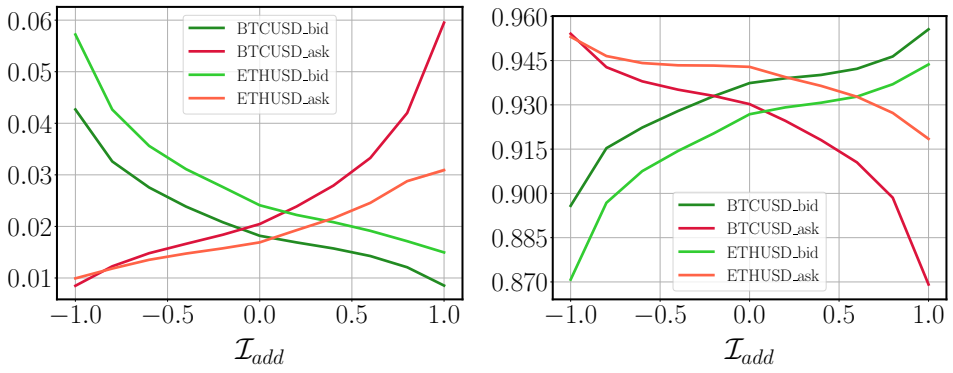
Let us now introduce three new variables and analyze the sensitivity of the fill probability and cancellation probability with respect to these variables. We use the Aalen-Johansen estimator defined in Equation (5), and 95% confidence intervals are computed using Equation (8).

3.3.1 Limit order flow imbalance

This measure quantifies the volume imbalance of limit orders that were posted in the last m events, right after the current order insertion, and indicates if the new bid/offer intentions are concentrated on one side of the LOB. Note that it is different from the order flow imbalance used in Su et al. (2021) and Cont et al. (2023) as cancellations and trades are not used for its computation. At the insertion of the order, we sum the volumes inserted on the bid side $\Delta_m Q_{\text{bid}}$ and on the ask side $\Delta_m Q_{\text{ask}}$ over the last m events (including this order). Notice that $\Delta_m Q_{\text{bid}} \geq 0$, $\Delta_m Q_{\text{ask}} \geq 0$ and $\Delta_m Q_{\text{bid}} + \Delta_m Q_{\text{ask}} > 0$ since the current posted limit order is taken into account. We thus define

$$\mathcal{I}^m_{\text{add}} := \frac{\Delta_m Q_{\text{bid}} - \Delta_m Q_{\text{ask}}}{\Delta_m Q_{\text{bid}} + \Delta_m Q_{\text{ask}}}. \tag{9}$$

To our knowledge, our work is the first to study the influence of this variable on fill probability. The number of events m is set to 50. The window size was chosen in order to obtain smooth monotonic probability functions. It would be straightforward to extend this indicator in order to give more weights to both relevant limits and more recent observations.



(a) Execution probability

(b) Cancellation probability

Figure 1: *Non-parametric analysis* — 1 second execution and cancellation probabilities as functions of the limit order flow imbalance $\mathcal{I}_{\text{add}}$ measured of the last 50 events.

The results for BTC-USD and ETH-USD are displayed in Figure 1. We observe that the fill probability is symmetrically monotonous as a function of the limit order flow imbalance, which is similar to the shape that one would obtain using the best queue imbalance. For the cancellation probability, we observe an inverse relationship, indicating that agents strongly condition their cancellation policy on measures of market depth variation. Note that we did not manage to obtain a similar empirical evidence for the equities, which suggests that the predictive power of this measure could vary from one asset type to another.

3.3.2 Aggressiveness index

For the purpose of analyzing the fill probability of aggressive order flow, *i.e.* orders that are posted inside the spread, we define a metric that will quantify the degree of aggressiveness of a newly inserted order. We therefore place ourselves in the case of a bid-ask spread before insertion that is (much) greater than 1 tick, limiting the scope of application of the new indicator to small-tick assets. We include the orders that are posted at touch, *i.e.* at the current best queue. In that case, they are passive but can be classified as aggressive orders with a zero aggressiveness index for practicality.

Let ψ be the bid-ask spread before the insertion of the order, and let δ be the distance of the order with respect to the best queue, such that $\delta > 0$ if the order is inserted inside the book, $\delta = 0$ if it is posted at

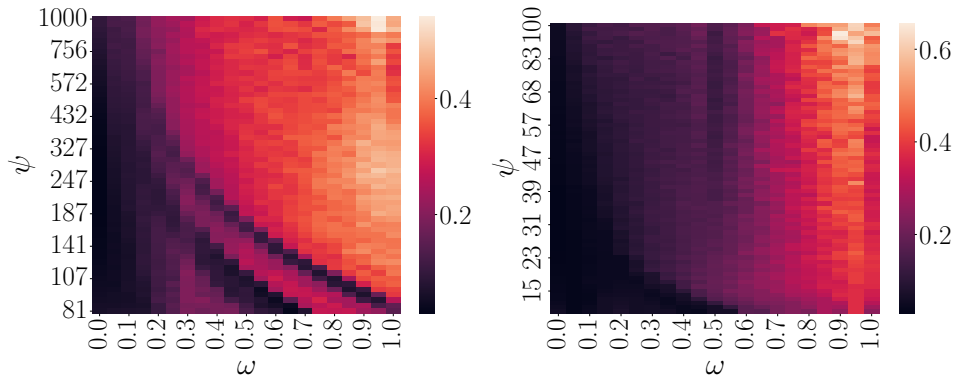
touch, and $-\psi < \delta \leq -1$ if it is aggressive. Both ψ and δ are expressed in number of ticks. We define the aggressiveness index as follows

$$\omega := \frac{\delta}{1-\psi}, \tag{10}$$

for $-\psi < \delta \leq -1$.

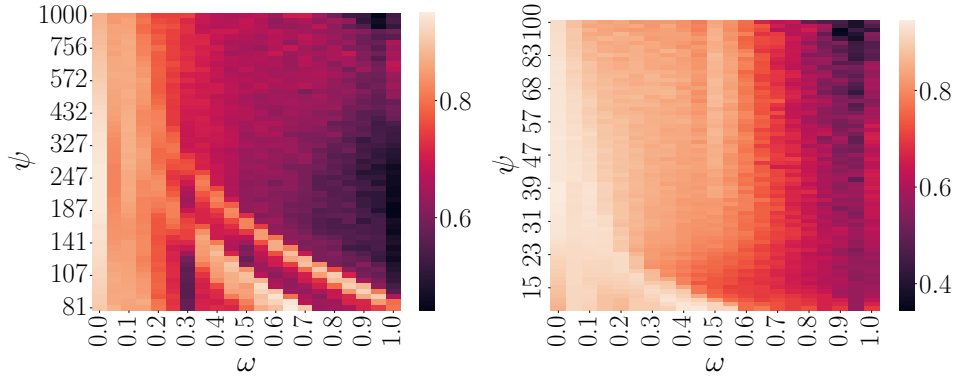
An index $\omega = 0$ corresponds to an order posted at the current best price while a value of 1 indicates a narrowing of the bid-ask spread to its minimal value, *i.e.*, 1 tick. This measure is also expressed in terms of the new bid-ask spread after insertion ψ^-, such that $\psi^- \leq \psi$, using the equality

$$\omega = \frac{\psi - \psi^-}{\psi - 1}. \tag{11}$$



(a) BTC-USD (b) ETH-USD

Figure 2: *Non-parametric analysis* — 1-second execution probability as a function of the aggressiveness index ω and the bid-ask spread ψ expressed in number of ticks.



(a) BTC-USD (b) ETH-USD

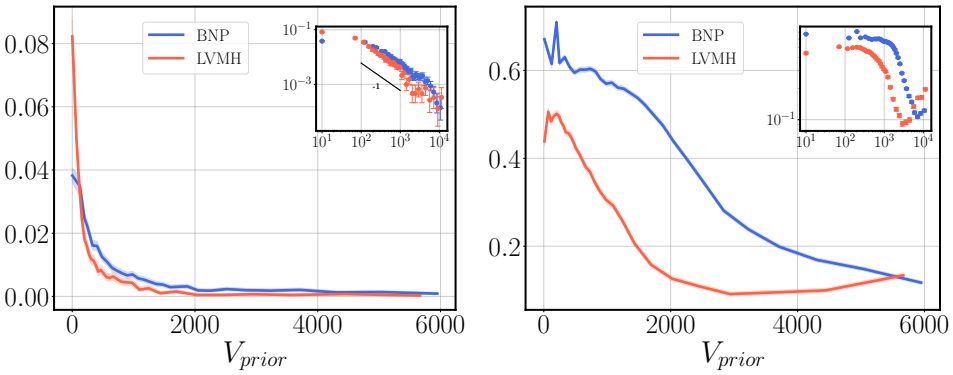
Figure 3: *Non-parametric analysis* — 1 second cancellation probability as a function of the aggressiveness index ω and the bid-ask spread ψ expressed in number of ticks.

When posting an order inside the bid-ask spread (and thus creating a new best queue), traders expect a higher execution probability and thus minimize the risk of non-execution while trading at a better price than if they had sent a market order and saving the taker fees. For small-tick assets, it is almost always possible to quote inside the spread and create a new best queue, as the bid-ask spread is generally greater than one tick. Therefore, traders who want to execute fast with minimum slippage may choose to place aggressive orders, leading to a significant tightening of the bid-ask spread as there many of them are competing at the same time. As outlined in Eisler et al. (2009), aggressive orders will cause an instantaneous rise in the intensity of liquidity taking and will rapidly become like any order resting in the best queue.

The results for the execution probability are shown in Figure 2, and those for the cancellation probability in Figure 3. We observe that the greater the aggressiveness index, the higher the fill probability, and the more aggressive the order is, the less likely it is to be cancelled. This emphasizes at least two phenomena: first, the propensity of impatient agents to optimize their price priority by placing their order at a better price than the current best limit. By doing so, a feedback effect happens: multiple orders are successively inserted in front of each other as the loss of price priority forces aggressive agents to cancel their order and place it again and so forth. The second one is the pinging activity in crypto venues, where orders are submitted inside the spread, at $\delta = -1$, and ca and cancelled shortly thereafter.

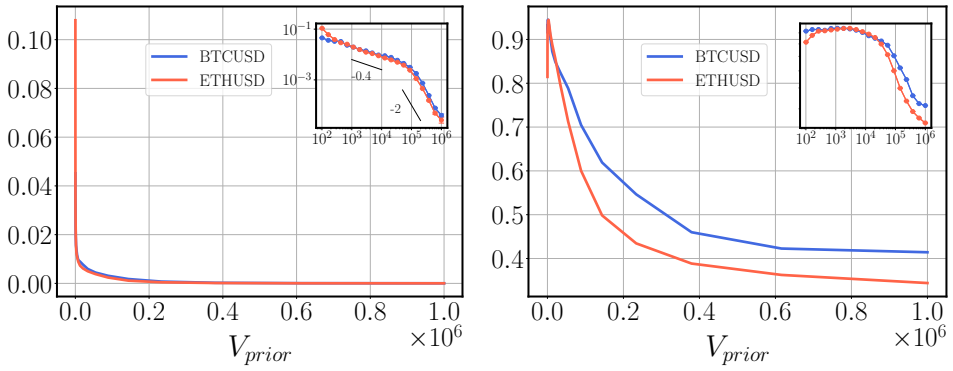
3.3.3 Priority volume

The priority volume, denoted by V_{prior} is computed by summing the volume of orders at better prices and those at the same price level with better time priority. This metrics complements the distance of placement of the limit order, denoted by δ. The smaller the priority volume, the greater the priority of execution when a marketable order hits the book.



(a) Execution probability (b) Cancellation probability

Figure 4: *Non-parametric analysis* — 10-second execution and cancellation probabilities as functions of the priority volume V_{prior} of the order. The volume is expressed in number of shares.



(a) Execution probability (b) Cancellation probability

Figure 5: *Non-parametric analysis* — 1-second execution and cancellation probabilities as functions of the priority volume V_{prior} of the order. The volume is expressed in USD.

Results are displayed in Figures 4 for the equities and 5 for the crypto pairs. Functions proportional to $V_{prior}^{-\alpha}$ with $\alpha \in \{0.4, 1, 2\}$ that appear as straight lines in a log-log plot are added for visual reference. Remarkably, the fill probability functions of the two equities are very similar, and those of the crypto pairs are too. For both asset classes, the probability decreases slowly with respect to the volume, but further investigations are needed to validate a specific parametric form. It is noteworthy that there is a major difference between both asset classes concerning the mechanisms that lie behind the execution of orders with respect to the prior pending liquidity. Indeed, small-tick limit order books are sparse, meaning that there are many gaps of liquidity within them; in other words, many prices are not quoted

in such LOBs, whereas the price limits of large tick assets are generally quoted up to some market depth. Thus, while the distance is a reasonable proxy for the price priority of an order in a large tick book, it can be misleading for small tick books as a large distance could be coupled with a small priority volume.

4 A fill probability model for tactical order placement

Our aim is to train an simple artificial neural network with well-chosen features and using raw data only. This makes it possible to use explainable AI to interpret the influence of each feature. We then apply it to optimal order placement and propose a backtest methodology to assess the performance of this type of strategy.

4.1 A note on the data used for training fill probability models

Our goal is to estimate from a feature vector Z the fill probability over a fixed time horizon T that we set to 1 second for crypto pairs and 10 seconds for equities. Our main assumption is that the order is not cancelled within the time horizon T. Nevertheless, instead of simply discarding the limit orders that are cancelled within the time horizon, we keep them for the loss weighting procedure that will be described later. Our procedure differs from Maglaras et al. (2022) in that we do not generate any synthetic order but rather train the model on historical order flow. Despite our exposition to selection bias, the benefit of this approach is threefold.

Firstly, as pointed out in Lo et al. (2002), fill probabilities of hypothetical limit orders do not lead to accurate estimates of the actual fill probability. While this is quite clear for market orders, posting a limit order also causes market response and price impact (see e.g. Eisler et al. (2012) for a consistent price impact analysis). Using a first passage time method would certainly inflate the true fill probability. Things get even worse when posting the order near the mid price since it modifies the liquidity imbalance, a key feature in next trade sign prediction. If computed on "infinitesimal" orders, using such a fill probability model with orders that may inverse the best liquidity imbalance would lead to biased results. It is hence of paramount importance to take this stylized fact into consideration when designing a fill probability model. The intuitive way of doing this is training the model on a data set of real limit orders. In practice, a trader could use her own trading history in order to take into consideration a cancellation tactics. Secondly, and most importantly, using real-life order flow allows one to build a model for aggressive limit orders, a task that is impossible to carry out when using a first passage time method. Last but not least, the raw size of the posted order becomes a key feature of the model as it obviously plays a significant role in a high-frequency setting, and the model can therefore learn the intricacies of its role over market's reaction.

4.2 Model training and feature importance

4.2.1 Training pipeline

We formulate the fill probability estimation problem as a binary classification problem with a suitable loss weighting procedure to account for censored data. For a data set with matrix representation $\mathbf{Z} := (z_{ij})_{1\leq i\leq N, 1\leq j\leq d}$ of N observations (rows) and d features (columns), we denote by $\mathbf{y} := (y_i)_{1\leq i\leq N}$ the vector of labels, where $y_i = 0$ indicates that the ith order was not executed under time horizon T and $y_i = 1$ indicates this order was filled. Since the time horizon of interest is small, we discarded limit orders that were posted too far in the book in order to remove noisy observations. The market depth threshold was fixed at 20 basis points of the mid price for the digital asset data base and 5 price limits for the equity one.

In Section 3, we have shown that the fill probability function presents smooth non-linear dependencies with respect to three new microstructure features, which makes the problem suitable for the training of neural networks. For this task, we use a feed-forward neural network with a sigmoid activation function for the output layer. We tested several architectures that all provided very similar results. For the results displayed in this work, we used 3 layers of 32 neurons with ReLU activation functions, and applied a 25% dropout for each layer in order to improve generalization. We add other interpretable variables to the set of features introduced in Section 3:

- distance of the order to the best queue;

- best queue imbalance, defined as follows: if q^b is the size of the best bid queue, and q^a the size of the best ask queue, then the best bid-offer (BBO) imbalance is $\frac{q^b - q^a}{q^b + q^a}$;
- size of the order;
- bid-ask spread;
- signed limit order flow, which is defined as $\Delta_m Q^b - \Delta_m Q^a$, using the notations of Section 3;
- signed order flow and order flow imbalance using both addition of liquidity (limit order insertion) and removal of liquidity (cancellation of a pending order or transaction), computed over the last 50 events;
- signed traded volume and imbalance of traded volume, computed over the last 50 transactions. We adopt the liquidity taker's viewpoint, *i.e.* if V_{bid} and V_{ask} are the traded volumes on the bid side and on the ask side over the last 50 transactions, then the signed traded volume is defined as $V_{\text{ask}} - V_{\text{bid}}$ and the traded volume imbalance as $\frac{V_{\text{ask}} - V_{\text{bid}}}{V_{\text{ask}} + V_{\text{bid}}}$;
- time elapsed since the last trade occurrence and the median duration of the last 50 trades, which could be characterized by the intensity of some self-exciting point process;
- volatility defined from a high-frequency estimator based on the uncertainty zone model of Robert and Rosenbaum (2011), and computed on traded prices of a moving window of 50 trades.

We rescale the variables using the classical Box-Cox transformation followed by a z-score. The hyperparameter of the Box-Cox transformation is chosen such that the statistic of the Kolmogorov-Smirnov test versus the standard Gaussian is minimized.

The significant presence of right censoring in the data set leads us to consider a loss weighting methodology. Simply discarding orders that are cancelled before the time horizon T from the data set would lead to a significant overestimation of the fill probability function. We use the "inverse-probability-of-censoring weighting" (IPCW) method to take care of this issue. This method is a well-known technique Mark and Robins (1993) and was successfully used in many real-world problems, see Vock et al. (2016) and Gonzalez Ginestet et al. (2021) for the details of the method and a comparative analysis. It was shown in Satten and Datta (2001) that an IPC-weighted version of the estimator of the survival function without censoring is equivalent to the Kaplan-Meier estimator, hence justifying the construction of this methodology. To our knowledge, our work is the first to apply IPCW to the training of a fill probability model.

Denote by w_i the weight associated to observation y_i, C_i the time of censoring which can be either a cancellation or other causes of censoring and E_i the time of execution. The IPC weights are defined as follows

$$w_i := \frac{\mathbb{1}_{\{\min(E_i, T) < C_i\}}}{\mathbb{P}\big(C_i > \min(E_i, T)\big)}. \tag{12}$$

The high-frequency activity occuring mainly near the mid price, we introduce a dependence of the weights to the distance of placement of limit orders. Such a modification will increase the weight applied to orders that were posted near best prices and even more for orders which manage to stay in the book until the time horizon T. Concerning orders that are posted inside the bid-ask spread, we propose to condition on the aggressiveness index as defined in Equation (11) to take into account the high cancellation rate of less aggressive orders. The censoring survival function is computed using the Kaplan-Meier estimator of Equation (4) but in this specific case, considering cancellation and right-censoring as death and execution as right-censoring.

The IPCW procedure deforms the fill probability in a similar way that the Kaplan-Meier function does by giving extra weights to orders that were not executed under the horizon or posted in highly censored configurations.

We focus on the bid side of BTC-USD and BNPP. We train the crypto model on 5 days — from 2022-11-11 to 2022-11-15 — and validate it on 2022-11-16. Note that we conducted the same test for other pairs and they all yielded similar results, which emphasizes the universality of the predictive power of the features we propose. For the equities, we train the model on 8 months —from January 1$^{\text{st}}$, 2017 to

August 31st, 2017— and validate it on 1.5 months —from September 1st, 2017 to October 15th, 2017. The much longer calendar duration in that case compensates for the much lower trading activity of equities.

4.2.2 Feature importance

We analyze the importance of features with Shapley values computed over out-of-sample observations using the SHAP library (Lundberg and Lee, 2017). The analysis is separated into three parts and brings insights about how the predictive power of features changes from order to order type, *i.e.* for aggressive, at-touch, and passive orders. Shapley diagrams are displayed in Figure 6.

1. **Passively posting ($\delta > 0$):** The distance and the order size are amongst the three most important features for both BTC-USD and BNPP. Interestingly the most important feature for the small tick asset is V_{prior}. Our understanding is that the distance alone is not sufficient to correctly characterize the priority in small tick order books because of their sparsity. For equal values of prior volume, different distances mean a different level of sparsity of the order book. If a small volume is quoted under a high distance δ, other market participants are likely to quote new prices at smaller distances than δ under the time horizon. Based on this thought, we believe both variables are inseparable when it comes to fill probability computation for small tick assets. The prior volume is also important for the equity but it comes after the order flow imbalance and the volatility.
2. **Posting at the current best ($\delta = 0$):** When posting at the current best queue, the bid-ask spread and the order size appear to be the most important features for both BTC-USD and BNPP. Interestingly, the importance of the BBO imbalance is smaller than that of the order flow imbalance for BNPP, which shows the importance of the dynamic features allowing the model to capture changes in the order flow.
3. **Aggressively posting ($-\psi < \delta < 0$):** When a new best queue is created, the bid-ask spread and the aggressiveness index are the most important features for the BTC-USD pair whereas for BNPP, the order size and the median trade duration are the most predictive ones. Since the spread of small tick assets is generally larger than one tick, the fill probability of an aggressive order is conditioned on the strength of the spread tightening it induces. Moreover, the limit order flow imbalance brings also an important contribution for the cryptocurrency pair, emphasizing the predictive power of this imbalance measure.

5 Application: optimal order placement

5.1 The cost function approach

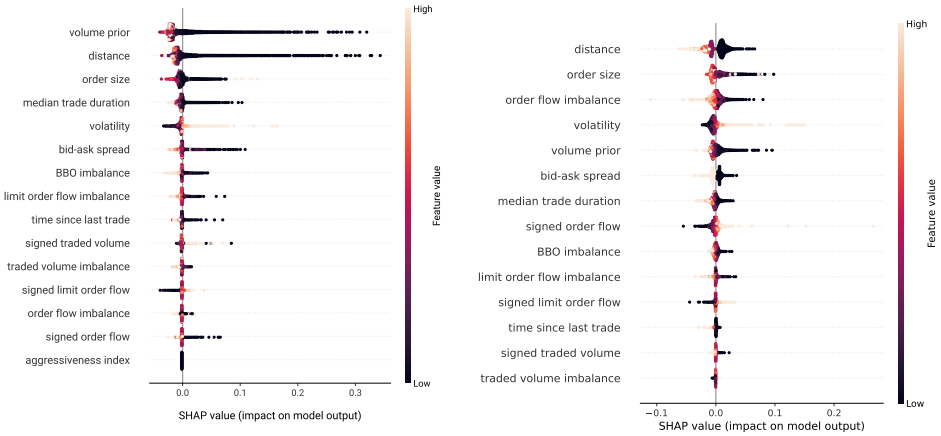
We concentrate on the tactical aspect of optimal trading. We consider an agent who aims at buying a small quantity q of an asset under a fixed horizon T that ranges from milliseconds to seconds. Here, a small quantity implies that the transaction volume will be managed by a single order, which can be either a market order or a limit order. We show that incorporating deep knowledge about the execution probability can enhance the decision-making process, and thereby improve performance.

5.1.1 Framework and notations

We place ourselves in the Implementation Shortfall optimization framework: the reference price of the execution algorithm is the initial mid price p_0 and the execution schedule must minimize the expected difference between the execution price (including fees) and this reference price. Before we detail the decision process, we need to introduce useful notations: denote by $\left(p_t^b\right)_t$, $\left(p_t^a\right)_t$, $(p_t)_t$, $(\psi_t)_t$ respectively the best bid price, the best ask price, the mid price, and the bid-ask spread processes of the asset. The tick size, expressed in quote units, is denoted by α, and we denote for $0 \leq t \leq T$ the variation of the best price $\bullet$ over $[0, t]$ by $\Delta p_t^\bullet := p_t^\bullet - p_0^\bullet$, for $\bullet \in \{b, a\}$. The agent observes a market state vector $z \in \mathbb{R}^d$.

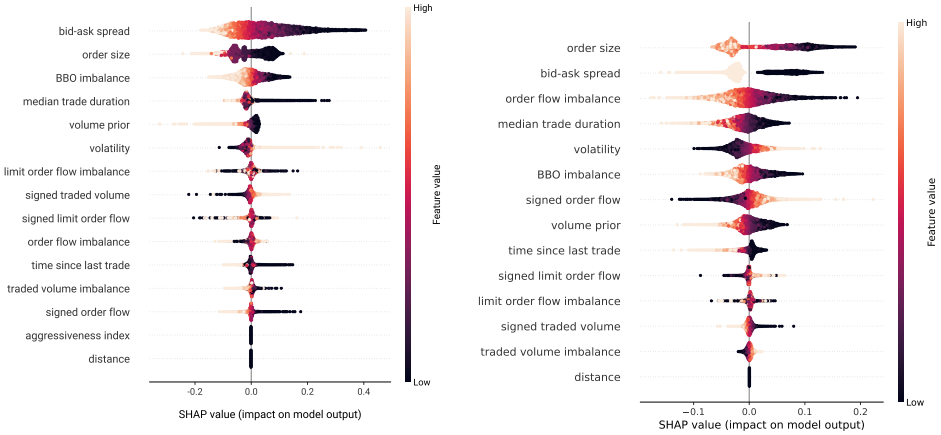
We denote by ε^- and ε^+ the taker and maker transaction fees respectively, such that $\varepsilon^- > \varepsilon^+$, and define the fee factors $f^- := 1 + \varepsilon^-$ and $f^+ := 1 + \varepsilon^+$. Note that in CEXs, the fixed transaction fees vary as a decreasing function of the traded volume which is often computed over a 30 days rolling window.

The agent has access to a limit order book and thus chooses between the two following tactics at the initial time 0.



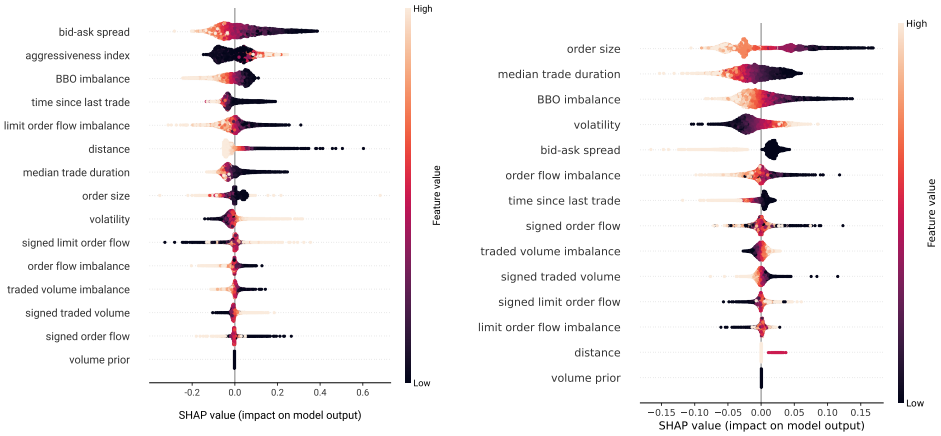
(a) Contributions for passive orders $\delta > 0$, BTC-USD

(b) Contributions for passive orders $\delta > 0$, BNPP



(c) Contribution at best quote $\delta = 0$, BTC-USD

(d) Contributions at best quote $\delta = 0$, BNPP



(e) Contributions for aggressive orders $\delta < 0$, BTC-USD

(f) Contributions for aggressive orders $\delta < 0$, BNPP

Figure 6: *Feature importance* — Market features contributions to the fill probability magnitude using Shapley values of 10,000 predictions, BTC-USD and BNPP, bid side.

- **Immediate execution tactic:** The agent crosses the bid-ask spread by sending a marketable order to get immediate execution at price p_0^a. The cost of this tactic is deterministic and will be denoted by $\mathcal{M}$. We suppose that q is sufficiently small such that the corresponding execution price is the best ask price p_0^a, *i.e.* there is no immediate market impact.
- **Post and wait tactic (PW):** The agent posts a buy limit order at bid price $p_0^b - \alpha\,\delta$ and waits for its execution until the time horizon T. The parameter δ is the distance to the best bid price as a number of ticks. The lifetime of this order will be denoted by $L^{\delta,q}$ following the conventions introduced in Section 3. Note that it is indexed by δ and q since it is associated to an order of size q placed at a distance δ to the best bid. We will denote by $F_T^{\delta,q,z} := \mathbb{P}\left(L^{\delta,q} \leq T | Z = z\right)$ its fill probability within time horizon T conditionally on a market state z. The cost function of the post and wait tactic is random and will be denoted by $\mathcal{W}(T, \delta, q)$.

As full execution is not guaranteed in the PW case, the agent will send a marketable order for immediate execution at the end of the period if the limit order is not filled by then, which will incur additional transaction costs in the case of adverse price moves. Both the fill probability and the clean-up cost increase with the volatility: the agent needs to find a trade-off between certainty of execution and management of the clean-up cost induced by market risk.

Henceforth, we remove the size factor q from the cost functions for the sake of clarity; in addition, $\mathbb{E}_z[.]$ corresponds to the conditional expectation $\mathbb{E}[.|z]$.

The expected execution costs of the two tactics are written as

$$\mathbb{E}_z\left[\mathcal{M}\right] = \mathcal{M} = f^- p_0^a - p_0, \tag{13}$$

and

$$\mathbb{E}_z\left[\mathcal{W}(T,\delta,q)\right] = F_T^{\delta,q,z}\left(f^+(p_0^b - \alpha\,\delta) - p_0\right) + (1 - F_T^{\delta,q,z})\,\mathbb{E}_z\left[\left(f^- p_T^a - p_0\right) | L^{\delta,q} > T\right]. \tag{14}$$

Note that the case of a partial execution is not taken into account since it is an unlikely occurence given that we consider small orders. Indeed, we checked empirically that a negligible proportion of orders were only partially executed under the time horizon T that we have chosen for the experiment.

The expected cost reduction if the agent chooses the post and wait tactics over immediate execution, denoted by $\mathcal{S}$, is defined as

$$\mathcal{S}(T,\delta,q,z) = \mathbb{E}_z\left[\mathcal{M} - \mathcal{W}(T,\delta,q)\right]. \tag{15}$$

Using elementary calculus, we find

$$\mathcal{S}(T,\delta,q,z) = \underbrace{F_T^{\delta,q,z}\left(f^- p_0^a - f^+(p_0^b - \alpha\delta)\right)}_{\textbf{(a)}} - \underbrace{(1 - F_T^{\delta,q,z})\,f^-\mathcal{V}_T^{\delta,q,z}}_{\textbf{(b)}}, \tag{16}$$

where

$$\mathcal{V}_T^{\delta,q,z} := \mathbb{E}_z\left[\Delta p_T^a \middle| L^{\delta,q} > T\right] \tag{17}$$

is the expected price variation of the best ask price over the period, conditionally on the non-execution of the pending order. To understand the role played by all variables better, we decomposed Equation (16) into two parts depending on the fate of the order:

- **(a)** if the limit order is fully executed, the agent saves the spread between the initial net of fees best ask price at which an immediate liquidity taking would have occurred and the net of fees bid price of the filled order;
- **(b)** in the case of a non execution, the agent incurs a transaction cost that may be greater than if an immediate execution had been chosen at the beginning. This clean-up cost is unknown at the beginning of the period and is characterized by the function $\mathcal{V}$.

The function $\mathcal{V}$ exhibits significant sensitivities to many variables such as realized volatility. This is intuitive, considering that the cleanup cost inherently reflects measures of volatility. To give additional intuition about the behaviour of this function, note that we expect it to behave as a non-increasing function of the total volume pending at better prices than the price of the posted order. Indeed, if the order is posted at the current best queue and does not get filled under the time period, it indicates that the market may have moved in the other direction thus inducing additional costs. This sensitivity is even stronger with aggressiveness. If the order is posted far from the best price, a non-execution does not necessarily indicate an adverse move of the opposite best price. Given all this remark, it is apparent that treating the clean-up cost as a constant in the objective of minimization would be simplistic.

Under suitable regularity conditions, the function $\mathcal{S}$ can be maximized over the set of admissible distances $\mathcal{A}_{\psi_0} := \{\delta \in \mathbb{Z},\, \delta > -\psi_0\}$ to find the optimal order placement strategy at fixed T and q, leading to the following optimization problem

$$\delta^* \;=\; \underset{\delta \in \mathcal{A}_{\psi_0}}{\operatorname{argmin}} \; -\mathcal{S}(T, \delta, q, z). \tag{18}$$

The unicity of the maximum is a challenge itself since we are dealing with highly non-linear dependencies that are inferred with a neural network. We therefore put that question aside for future investigation.

5.1.2 A toy model

Before diving into the estimation of $\mathcal{V}$, let us introduce a simplified version of the order placement model with an exponential fill probability function as specified in Avellaneda and Stoikov (2008), Laruelle et al. (2013). For the sake of clarity, we set $f^- = f^+ = 1$ and we consider the modified distance $\delta^a := \psi_0 + \delta$ of the order the best ask price at time 0, expressed in ticks. We set $\mathcal{V}$ constant and for all $\delta^a \geq 1$,

$$F^\delta = Ae^{-k\delta^a}. \tag{19}$$

The expected saved cost function writes:

$$\mathcal{S}(\delta) = Ae^{-k\delta^a}\delta^a - \left(1 - Ae^{-k\delta^a}\right)\mathcal{V}. \tag{20}$$

Setting the condition $k(1+\mathcal{V}) \leq 1$ and differentiating with respect to δ^a, we obtain the following optimal distance of placement:

$$\delta^{a,*} := \frac{1}{k} - \mathcal{V} \tag{21}$$

and the associated maximum of the saved cost function:

$$\mathcal{S}(\delta^{a,*}) = \frac{A}{k}e^{k\mathcal{V}-1} - \mathcal{V}. \tag{22}$$

Equation 21 indicates that in this simple framework, the optimal distance should scale linearly with respect to the expected adverse price move $\mathcal{V}$, and inversely with respect to the decay rate of the fill probability. This model will later be used as a benchmark for assessing the performance of the full execution algorithm.

5.1.3 The effect of CEXs fee policy on the strategy

When trading in CEXs, agents may face very different transaction fees depending on their monthly turnover. We build an example to illustrate the strong sensitivity of the strategy to the fee policy.

We consider an agent who posts a buy limit order in the book at the current best bid price $p_0^b = 19,999.50$ USD and a bid ask spread of 1.00 USD. Suppose that the model predicts a clean-up cost $\mathcal{V} = 2.00$ USD (which would correspond to an annualized volatility of approximately 56%). We present the trading fee policy of Coinbase in Table 3. In Figure 7, we display the decision map of the execution algorithm as a function of the fee level and the execution probability. We clearly observe that for a fixed execution probability, the fee level has a strong impact over the optimal decision.

Table 3: *Trading fees* — Coinbase fee policy of spot trading on April 2023.

Level	**Monthly turnover**	ε^-	ε^+
1	[\$0, \$10K)	0.006	0.004
2	[\$10K, \$50K)	0.004	0.0025
3	[\$50K, \$100K)	0.0025	0.0015
4	[\$100K, \$1M)	0.002	0.001
5	[\$1M, \$15M)	0.0018	0.0008
6	[\$15M, \$75M)	0.0016	0.0006
7	[\$75M, \$250M)	0.0012	0.0003
8	[\$250M, \$400M)	0.0008	0
9	[\$400M, ∞)	0.0005	0

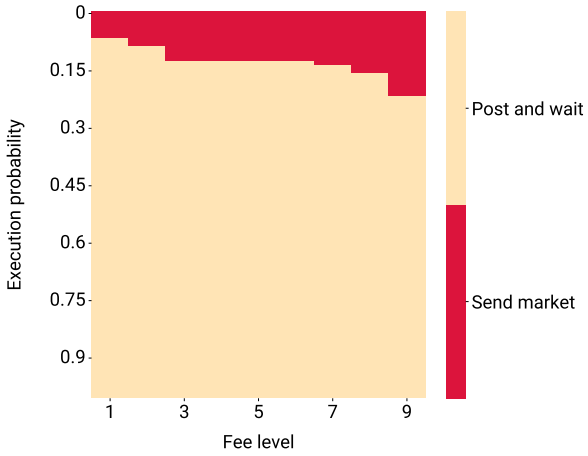


Figure 7: *Decision boundary* — Decision map in the practical example as a function of fee level and execution probability. Fee levels are displayed in Table 3.

5.1.4 Estimation of the expected adverse price move $\mathcal{V}$

We propose a simple methodology to estimate the clean-up cost function $\mathcal{V}$. Using the level 3 data we track the best ask price dynamics after each order's insertion and record its variation over the time window $[0, T]$ when the order is not executed. We remove the orders that were cancelled, executed or censored before T since our only interest is in the events $\{L^{\delta,q} > T\}$. By doing so, we are able to build feature buckets and compute the average price move per bucket. An example of the shape for the estimator of $\mathcal{V}$ with respect to the realized volatility σ is presented in Figure 8. We observe a smooth dependence, indicating that the function $\mathcal{V}$ shares analogous properties with the fill probability function and can be estimated using a neural network model, and, once again, with a simple architecture. This measure is closely related to the volatility, which makes it much more predictable than the raw price moves.

We train a neural network with the same architecture as the one of the fill probability model, but with a linear activation function in the output layer. Using both trained NNs, the saved cost function $\mathcal{S}$ is finally estimated using Equation (16). Further analysis of the importance of each variable in the decision making process, can be found in Appendix 6, Figure 12. In addition to the key features that we described for the fill probability function, the limit order flow imbalance turns out to play a major role for cryptos in the estimation of the saved cost function in the three considered configurations: passively posting, posting at the current best queue and aggressively posting.

5.2 Backtest of the order placement router

Historical backtests are often misleading for many practical reasons such as the absence of market impact that would be undoubtedly caused by the algorithm on a real market. For example, the insertion of a new limit order in the first queue would adversely modify the best queues imbalance and negatively impact the strategy execution outcome. Considering infinitesimal sizes for orders does not help that

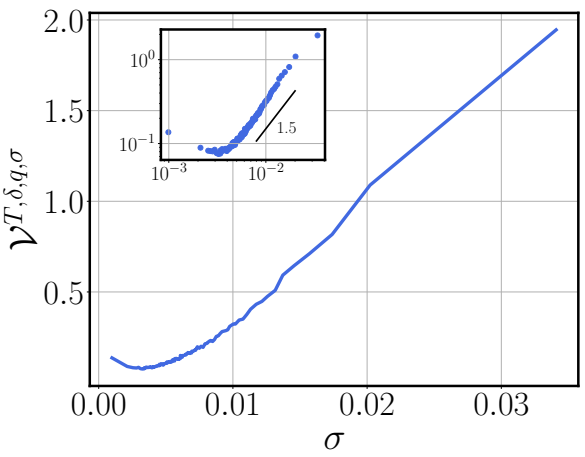


Figure 8: *The clean-up cost as a measure of market risk* — Empirical estimate of $\mathcal{V}$, in USD, as a function of realized high-frequency volatility σ, expressed in %/trade, for BTC-USD.

much considering it will not be the case once the algorithm is sent to production. This is even more true for aggressive orders since they cause an immediate spread narrowing, thus inducing much more reaction from the market. The absence of market impact inflates the true out of sample performance, and another solution needs to be proposed for high frequency execution algorithms.

5.2.1 Towards an impact-adjusted backtest

We select real limit orders that were posted in the book and compute their expected saved costs $\mathcal{S}$. Each order should either be filled under the horizon T or left in the book at least for a time T. The characteristics of the selected orders should reflect those of the orders that are posted by the strategy. For example, if our execution tactic posts orders with sizes ranging from 100 USD to 1,000 USD, and with distances smaller than 5 basis points, then the data set should be composed of orders with respect to these constraints. For each limit order in this test set, the sign of $\mathcal{S}$ will indicate whether the execution algorithm would have effectively posted the order, *i.e.* $\mathcal{S} > 0$, or opted for immediate execution instead, *i.e.* , *i.e.* $\mathcal{S} < 0$. This step leads to an hypothetical decision $d = \mathbb{1}_{\mathcal{S}>0}$. Then, one of the following three outcomes is observed at the time horizon T.

1. The limit order was executed. The true optimal decision is $\hat{d} = 1$.
2. The limit order was not filled under T, but the best ask price variation is negative, hence leading to an improved execution price. The true optimal decision is $\hat{d} = 1$.
3. The limit order was not filled under T, and the best ask price variation is positive, hence leading to extra transaction costs. The true optimal decision is $\hat{d} = 0$.

This labelling procedure enables us to deduce binary classification metrics to assess the performance of the model in making the right decision, by comparing the predicted d with the observed $\hat{d}$. It is important to note that by proceeding so, we are able to test the decision-making algorithm, but not the effectiveness of the optimal distance of insertion.

The backtest procedure is illustrated in the diagram of Figure 9.

5.2.2 Experiment setting

For the CEXs data base, both fill probability and clean-up cost models are trained on 5 days, validated on two days, and tested on the following week, representing 3 test periods over the month. The equity model is trained on 8 months, validated on 1.5 months, and tested on 2.5 months.

We now describe the three models that will be tested and compared below. The chosen fee policy for the crypto trading algorithm is the level 9, *i.e.* $\varepsilon^- = 5$ basis points, $\varepsilon^+ = 0$.

- **Model I:** The toy model introduced in 5.1.2 provides a closed form expression of the saved cost for any distance of placement δ, see Equation (20). The fill probability parameters A and k are

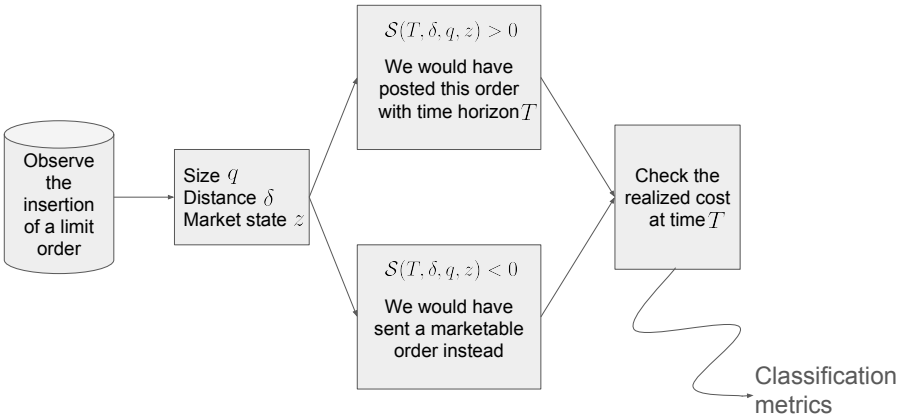


Figure 9: *Impact-adjusted backtest procedure* — Diagram of the methodology for performance evaluation of a tactical execution algorithm.

estimated by fitting an exponential form on the Kaplan-Meier function considering cancellation as right-censoring and $\mathcal{V}$ is computed as the average of best ask price moves observed at horizon for every limit orders in the training set that are not executed. A more sophisticated estimation procedure could be adapted, for example by taking into account the intraday seasonality of market activity.

- **Model II:** The fill probability function used for this benchmark is the neural network model described in Section 4, and $\mathcal{V}$ is a constant computed as in the exponential toy model benchmark. Thus, the only difference with the full model resides in the estimation of $\mathcal{V}$.
- **Model III:** Combination of the two neural network models for the fill probability and the expected best price move at horizon.

The results are displayed in Table 4, which reports the respective performance of the three models for BTC-USD and BNPP. We observe a clear improvement of the decision-making process with the use of handcrafted features and non-linear models. It demonstrates that a simple and interpretable neural network architecture fed with well-chosen features may be sufficient to design decent tactical execution algorithms. Furthermore, we observe that using a state-dependent market risk $\mathcal{V}$ instead of a constant one seems to be crucial, as the F-score for both asset classes drastically improves. Interestingly, the algorithm performs much better on BTC-USD than on the BNPP stock. One possible interpretation is that BTC-USD is more predictable than BNPP.

Table 4: *Impact-adjusted backtest* — Performance metrics of the backtest of three order placement trading engines

Asset		Precision	Recall	F-score
BTC-USD	I	0.21	0.99	0.34
	II	0.27	0.66	0.38
	III	0.37	0.81	**0.51**
BNPP	I	0.11	0.05	0.07
	II	0.10	0.21	0.14
	III	0.20	0.54	**0.30**

5.2.3 Analysis of the optimal distance for small tick assets

Using model III, we compute the optimal distance of placement for BTC-USD and illustrate the behaviour of the algorithm with two cases. The first one is displayed in Figure 10 and represents a heatmap of the

expected saved cost $\mathcal{S}$ as a function of the bid-ask spread and the distance of placement, with the Level 9 fee policy corresponding to 5 bps of taker fees and no maker fees. The heatmap is computed using features that are observed at a random point in time. We observe that the algorithm tends to be aggressive and to quote inside the spread. It indicates that it exclusively focuses on maximizing the fill probability and saving the taker fees, even though the best way to do it is to narrow the spread and reduce the price discount of the limit order. The second illustration is displayed in Figure 11 and is also a heatmap of the expected saved cost $\mathcal{S}$ computed at the same point in time as the previous one, but assuming there are no fees. We observe that the algorithm quotes deeper in the book than in the previous case. To wrap this up, the algorithm tends to post aggressive orders when the taker - maker fee gap $\varepsilon^- - \varepsilon^+$ is large, pushing the optimal distance towards $-\psi$ as this gap increases. This result is particularly interesting as it demonstrates that CEXs fee policies tend to push agents to be more aggressive when it comes to executing fast. We believe this brings new elements of understanding of the impact of the fee policy on trading activity and agents behaviours in CEXs. Finally, we see that the optimal distance of placement given by the full model seems to be linear or at least sub-linear in the bid-ask spread, which is consistent with the prediction of the toy model we introduced.

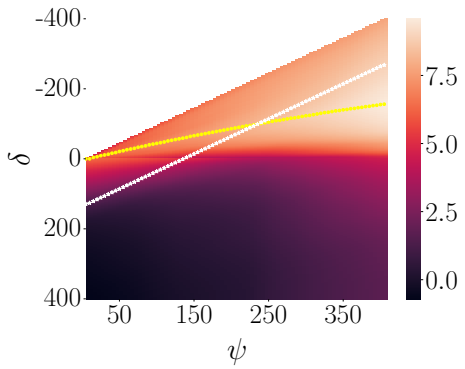


Figure 10: *Optimal distance of placement* — Example of the optimal distance policy δ^*, as a function of the bid-ask spread ψ, BTC-USD setting fees to the level 9 of Table 3. Both the distance δ and the bid-ask spread ψ are expressed in number of ticks. The color bar represents the values taken by the expected saved cost $\mathcal{S}$. The optimal distance δ^* of model I given by Equation (21) is represented by a white $\star$ and the optimal distance from model III is represented by a yellow $\circ$.

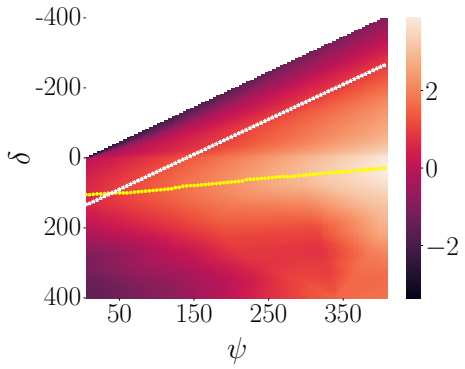


Figure 11: *Optimal distance of placement* — Example of the optimal distance policy δ^* as a function of the bid-ask spread ψ, BTC-USD, setting the fees to zero. Both the distance δ and the bid-ask spread ψ are expressed in number of ticks. The heatmap represents the values taken by the expected saved cost $\mathcal{S}$. The optimal distance δ^* of model I given by Equation (21) is represented by a white $\star$ and the optimal distance from model III is represented by a yellow $\circ$.

5.2.4 Penalizing aggressiveness with latency risk: a practitioner viewpoint

In the case of small tick assets, we observed the resulting optimal order placement policy can place limit orders inside the spread. While posting a limit order in the spread guarantees a higher execution

probability, this action brings a risk to the table: the instantaneous volatility of the best opposite price. Let us go back to the framework in which an agent is willing to buy a certain amount of a small tick asset. If the algorithm inserts an order inside the spread, there is no guarantee that between the moment the update message is sent by the exchange and the time the optimal distance is computed and the resulting order is sent, *i.e.* which is commonly called the tick-to-trade latency, other agents have not quoted a new ask price inside the spread too. Hence, crossing this new best ask price would generate additional transaction costs since the limit order would instantly become a marketable order, incurring taker fees. This latency risk should be integrated in the cost function of the trading algorithm in order to penalize extreme aggressiveness.

To provide food for thought about modeling such a risk in our framework, let us denote by $\ell > 0$ the tick-to-quote latency of the algorithm, *i.e.* the time in seconds that separates the moment the message is sent by the venue, processed by the matching engine and the time the order resulting from the execution tactics is posted in the LOB. We suppose in the rest of the discussion that ℓ is negligible compared to the time horizon T, for example $\frac{\ell}{T} < 10^{-3}$. This condition is not only necessary because the performance of high-frequency strategies vanishes with respect to latency, but is also convenient because we can assume

$$\Delta p^a_\ell \perp\!\!\!\perp L^{\delta,q}, \tag{23}$$
$$\Delta p^a_\ell \perp\!\!\!\perp \Delta p^a_T, \tag{24}$$

where $\square \perp\!\!\!\perp \triangle$ stands for "$\square$ and $\triangle$ are independent random variables".

We now denote by $\phi^z_\ell(x) := \mathbb{P}\left(\Delta p^a_\ell \leq x | Z = z\right)$ for a real number x the cumulative distribution function of the best ask price variation over $[0, \ell]$ conditionally on a market state z. The latency-sensitive saved cost function $\mathcal{S}_\ell$ of the post and wait tactic is now written as

$$\begin{aligned}\mathcal{S}_\ell(T, \delta, q, z) &= \left(1 - \phi^z_\ell\left(-(\psi_0 + \delta)\right)\right)\mathcal{S}(T, \delta, q, z) \\ &- \phi^z_\ell(-(\psi_0 + \delta)) f^- \mathbb{E}_z\left[\Delta p^a_\ell | \Delta p^a_\ell \leq -(\psi_0 + \delta)\right],\end{aligned} \tag{25}$$

with $\mathcal{S}$ being given in Equation (16).

The last term represents the price discount of the marketable limit order over the immediate execution tactics due to a best ask price improvement. We see that the value of this new expected saved cost function is not necessarily smaller than the value of the latency-free one $\mathcal{S}$ but the maximum of the function is possibly attained at a different distance δ.

6 Discussion and conclusion

In this work we introduced new microstructural features for fill probability computation, namely, the limit order flow imbalance, the aggressiveness index and the priority volume. We demonstrated their predictive power by exploring the smooth dependence of the fill and cancellation probability functions with respect to these features. We showed how neural networks with simple architectures can be used for the fill probability and clean-up cost computation using high-frequency data. A neural network was trained on a data base of real limit orders using a set of handcrafted interpretable features, and we analyzed the differences in the feature importance between CEXs cryptocurrency pairs and Euronext equities. We explained how taking into account both the priority volume and the distance of placement may be crucial in the case of small tick cryptocurrencies due to the sparse nature of their order book. Concerning the use of real limit orders over hypothetical orders, we discussed the main advantage in exploiting such informative data and strongly suggested to favor the real order flow other synthetic orders to eliminate the zero market impact assumption. By designing a cost function for an agent who aims at buying a quantity of the asset within a short time horizon, we demonstrated how to integrate such a model in a trading engine. A new backtest method was proposed, allowing to account for the market impact of limit orders using historical data only. We assessed the performance of the model and compared it to a toy model that involves a common form of the fill probability function. By computing the decisions the model would have made at the insertion of real limit orders, classification metrics can be used to study the relevancy of these execution models in the decision-making process. Finally, examples of optimal distance of placement were provided in the case of a small tick asset and numerical experiments suggest that the fee policy of the trading venue plays a decisive role concerning the aggressiveness of the order. Our findings suggest that in CEXs, provided that there is no latency, posting extremely aggressive limit

orders is often optimal. The efficiency of such a radical tactic is explained by two main factors, the first one being the high fill probability of such an order and the second one being the fee policy that sets a significant difference between the maker fee and the taker fee. Hypothetically, inserting a bid limit order 1 tick below the best ask price when the spread is of the order of several hundreds of ticks would almost surely save 5 basis points of trading costs. In practice, such a tactic is prone to latency risk, and it can be integrated in the computation of the expected cost function.

In a multi-horizon framework, *e.g.* execution algorithms with a trading horizon that can range from milliseconds to minutes depending on the market regime or operational constraints, the inference of the whole survival function is necessary. In this case, estimating more sophisticated models such as the ones from survival deep learning literature or the convolutional-transformer of Arroyo et al. (2024) would be relevant. But more work needs to be done in order to propose an architecture that allows for fast computations in a live trading environment (computation time $<$ 1 microsecond). Another extension of our approach would be to train a model for post-insertion evaluation. In a nutshell, the insertion of a limit order in the book bumps the order flow intensity and this excitation vanishes over time. This causes the fill probability of an order at its insertion to differ from the fill probability of a pending order with the exact same characteristics and the exact same set of features. The insertion of liquidity reveals information about the agent's intention and impacts the price, which is likely to move in the opposite direction. Such an extension can be carried out by simply adding pending limit orders in the training data and creating a new feature that characterizes the time elapsed since their insertion in the book. Last but not least, the study of the convexity of the saved cost function would highlight some regularity conditions that can be added as non-linear constraints in the loss functions of the neural network in order to improve the computation of the optimal distance.

References

Aalen, O. (1976). Nonparametric inference in connection with multiple decrement models. *Scandinavian Journal of Statistics*, pages 15–27.

Aalen, O. (1978). Nonparametric inference for a family of counting processes. *The Annals of Statistics*, pages 701–726.

Aalen, O. and Johansen, S. (1978). An empirical transition matrix for non-homogeneous markov chains based on censored observations. *Scandinavian Journal of Statistics*, pages 141–150.

Arroyo, A., Cartea, A., Moreno-Pino, F., and Zohren, S. (2024). Deep attentive survival analysis in limit order books: Estimating fill probabilities with convolutional-transformers. *Quantitative Finance*, 24(1):35–57.

Avellaneda, M. and Stoikov, S. (2008). High-frequency trading in a limit order book. *Quantitative Finance*, 8(3):217–224.

Bacry, E., Jaisson, T., and Muzy, J.-F. (2016). Estimation of slowly decreasing hawkes kernels: application to high-frequency order book dynamics. *Quantitative Finance*, 16(8):1179–1201.

Bayraktar, E. and Ludkovski, M. (2014). Liquidation in limit order books with controlled intensity. *Mathematical Finance*, 24(4):627–650.

Braun, T. M. and Yuan, Z. (2007). Comparing the small sample performance of several variance estimators under competing risks. *Statistics in medicine*, 26(5):1170–1180.

Brogaard, J., Hendershott, T., and Riordan, R. (2019). Price discovery without trading: Evidence from limit orders. *The Journal of Finance*, 74(4):1621–1658.

Cartea, A. and Jaimungal, S. (2015). Optimal execution with limit and market orders. *Quantitative Finance*, 15(8):1279–1291.

Cho, J.-W. and Nelling, E. (2000). The probability of limit-order execution. *Financial Analysts Journal*, 56(5):28–33.

Cont, R., Cucuringu, M., and Zhang, C. (2023). Cross-impact of order flow imbalance in equity markets. *Quantitative Finance*, 23(10):1373–1393.

Cont, R. and Kukanov, A. (2017). Optimal order placement in limit order markets. *Quantitative Finance*, 17(1):21–39.

Donnelly, R. and Gan, L. (2018). Optimal decisions in a time priority queue. *Applied Mathematical Finance*, 25(2):107–147.

Eisler, Z., Bouchaud, J.-P., and Kockelkoren, J. (2012). The price impact of order book events: market orders, limit orders and cancellations. *Quantitative Finance*, 12(9):1395–1419.

Eisler, Z., Kertesz, J., Lillo, F., and Mantegna, R. N. (2009). Diffusive behavior and the modeling of characteristic times in limit order executions. *Quantitative Finance*, 9(5):547–563.

Gonzalez Ginestet, P., Kotalik, A., Vock, D. M., Wolfson, J., and Gabriel, E. E. (2021). Stacked inverse probability of censoring weighted bagging: A case study in the infcarehiv register. *Journal of the Royal Statistical Society Series C: Applied Statistics*, 70(1):51–65.

Hautsch, N. and Huang, R. (2012). The market impact of a limit order. *Journal of Economic Dynamics and Control*, 36(4):501–522.

Huang, W., Lehalle, C.-A., and Rosenbaum, M. (2016). How to predict the consequences of a tick value change? evidence from the tokyo stock exchange pilot program. *Market Microstructure and Liquidity*, 2(03n04):1750001.

Kalbfleisch, J. D. and Prentice, R. L. (2011). *The statistical analysis of failure time data*. John Wiley & Sons.

Kaplan, E. L. and Meier, P. (1958). Nonparametric estimation from incomplete observations. *Journal of the American statistical association*, 53(282):457–481.

Katzman, J. L., Shaham, U., Cloninger, A., Bates, J., Jiang, T., and Kluger, Y. (2018). Deepsurv: personalized treatment recommender system using a cox proportional hazards deep neural network. *BMC medical research methodology*, 18:1–12.

Laruelle, S., Lehalle, C.-A., et al. (2013). Optimal posting price of limit orders: learning by trading. *Mathematics and Financial Economics*, 7(3):359–403.

Laruelle, S., Rosenbaum, M., and Savku, E. (2019). Assessing mifid ii regulation on tick sizes: A transaction costs analysis viewpoint. *Market Microstructure and Liquidity*, 5(01n04):2050003.

Lee, C., Zame, W., Yoon, J., and Van Der Schaar, M. (2018). Deephit: A deep learning approach to survival analysis with competing risks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32.

Lehalle, C.-A. and Mounjid, O. (2017). Limit order strategic placement with adverse selection risk and the role of latency. *Market Microstructure and Liquidity*, 3(01):1750009.

Lo, A. W., MacKinlay, A. C., and Zhang, J. (2002). Econometric models of limit-order executions. *Journal of Financial Economics*, 65(1):31–71.

Lundberg, S. M. and Lee, S.-I. (2017). A unified approach to interpreting model predictions. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30*, pages 4765–4774. Curran Associates, Inc.

Maglaras, C., Moallemi, C. C., and Wang, M. (2022). A deep learning approach to estimating fill probabilities in a limit order book. *Quantitative Finance*, 22(11):1989–2003.

Mark, S. D. and Robins, J. M. (1993). A method for the analysis of randomized trials with compliance information: an application to the multiple risk factor intervention trial. *Controlled clinical trials*, 14(2):79–97.

Markov, V. (2014). On the design of sell-side limit and market order tactics. *arXiv preprint arXiv:1409.1442*.

Moallemi, C. C. and Yuan, K. (2016). A model for queue position valuation in a limit order book. *Columbia Business School Research Paper*.

Nelson, W. (1969). Hazard plotting for incomplete failure data. *Journal of Quality Technology*, 1(1):27–52.

Nelson, W. (1972). Theory and applications of hazard plotting for censored failure data. *Technometrics*, 14(4):945–966.

Perelló, J., Gutiérrez-Roig, M., and Masoliver, J. (2011). Scaling properties and universality of first-passage-time probabilities in financial markets. *Physical Review E*, 84(6):066110.

Pintilie, M. (2006). *Competing risks: a practical perspective*. John Wiley & Sons.

Robert, C. Y. and Rosenbaum, M. (2011). A new approach for the dynamics of ultra-high-frequency data: The model with uncertainty zones. *Journal of Financial Econometrics*, 9(2):344–366.

Said, E., Ayed, A. B. H., Husson, A., and Abergel, F. (2017). Market impact: A systematic study of limit orders. *Market microstructure and liquidity*, 3(03n04):1850008.

Satten, G. A. and Datta, S. (2001). The kaplan–meier estimator as an inverse-probability-of-censoring weighted average. *The American Statistician*, 55(3):207–210.

Su, Y., Sun, Z., Li, J., and Yuan, X. (2021). The price impact of generalized order flow imbalance. *arXiv preprint arXiv:2112.02947*.

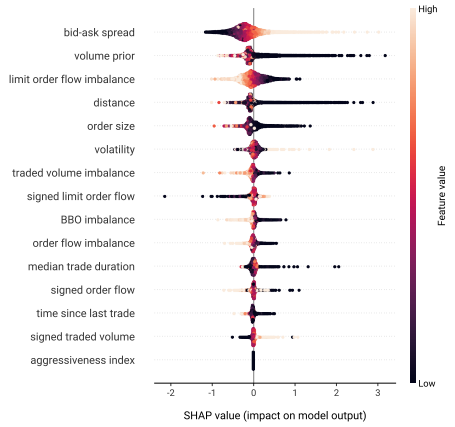
Vock, D. M., Wolfson, J., Bandyopadhyay, S., Adomavicius, G., Johnson, P. E., Vazquez-Benitez, G., and O'Connor, P. J. (2016). Adapting machine learning techniques to censored time-to-event health record data: A general-purpose approach using inverse probability of censoring weighting. *Journal of biomedical informatics*, 61:119–131.

Wald, J. K. and Horrigan, H. T. (2005). Optimal limit order choice. *The Journal of Business*, 78(2):597–620.

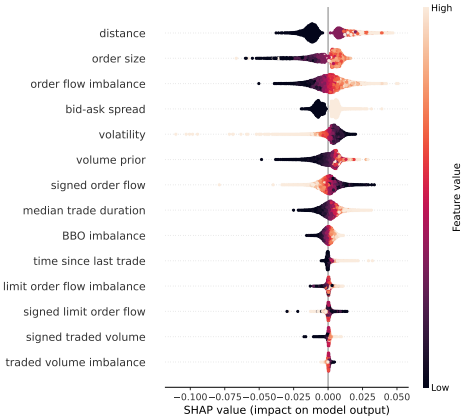
Wang, Z. and Sun, J. (2022). Survtrace: Transformers for survival analysis with competing events. In *Proceedings of the 13th ACM international conference on bioinformatics, computational biology and health informatics*, pages 1–9.

Weber, P. and Rosenow, B. (2005). Order book approach to price impact. *Quantitative Finance*, 5(4):357–364.

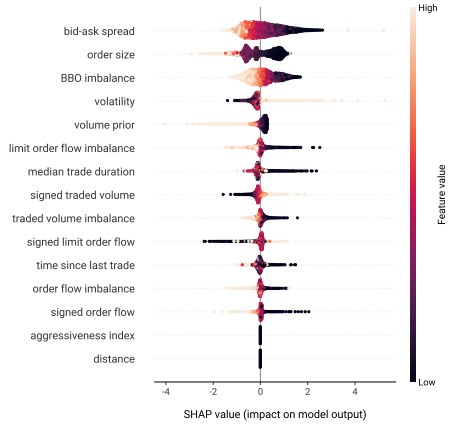
Appendix



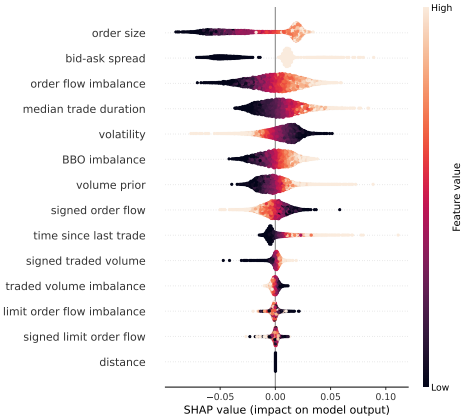
(a) Contributions for passive orders $\delta > 0$, BTC-USD



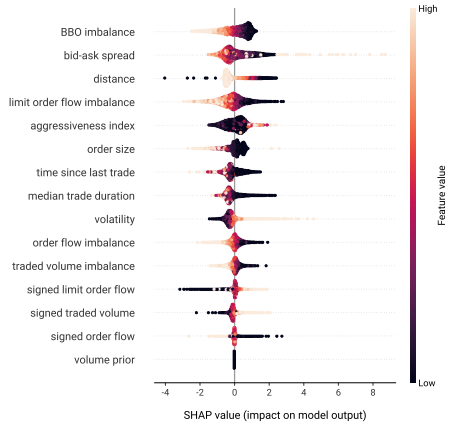
(b) Contributions for passive orders $\delta > 0$, BNPP



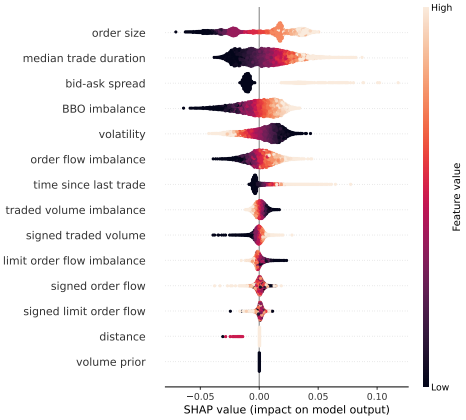
(c) Contribution at best quote $\delta = 0$, BTC-USD



(d) Contributions at best quote $\delta = 0$, BNPP



(e) Contributions for aggressive orders $\delta < 0$, BTC-USD



(f) Contributions for aggressive orders $\delta < 0$, BNPP

Figure 12: *Feature importance* — Market variables contributions to the saved cost function magnitude using Shapley values of 10,000 predictions, BTC-USD and BNPP, bid side.

978-1-5386-5541-2/18/$31.00 ©2018 IEEE

This paper is a preprint version of the paper:
Gašperov, Bruno, and Zvonko Kostanjčar. "Deep Reinforcement Learning for Market Making Under a Hawkes Process-Based Limit Order Book Model." *IEEE Control Systems Letters* 6 (2022): 2485-2490, DOI: 10.1109/LCSYS.2022.3166446, https://ieeexplore.ieee.org/document/9754690 and it is IEEE copyrighted material. For IEEE copyright policy please refer to https://www.ieee.org/publications/rights/copyright-policy.html

arXiv:2207.09951v1 [q-fin.GN] 20 Jul 2022

Deep Reinforcement Learning for Market Making Under a Hawkes Process-Based Limit Order Book Model

Bruno Gašperov and Zvonko Kostanjčar, *Member, IEEE*

***Abstract*— The stochastic control problem of optimal market making is among the central problems in quantitative finance. In this paper, a deep reinforcement learning-based controller is trained on a weakly consistent, multivariate Hawkes process-based limit order book simulator to obtain market making controls. The proposed approach leverages the advantages of Monte Carlo backtesting and contributes to the line of research on market making under weakly consistent limit order book models. The ensuing deep reinforcement learning controller is compared to multiple market making benchmarks, with the results indicating its superior performance with respect to various risk-reward metrics, even under significant transaction costs.**

***Index Terms*— Finance, neural networks, stochastic optimal control.**

I. Introduction

OPTIMAL market making (MM) is the problem of placing bid (buy) and ask (sell) orders simultaneously on both sides of the limit order book (LOB) with the goal of maximizing trader's terminal wealth, all while minimizing the associated risks. Arguably the most salient among them is the inventory risk, which stems from price movements in the underlying asset held by the trader (market maker) in its inventory. A risk-averse market maker typically prefers to keep its inventory consistently close to zero. In order to achieve this, continuous adjustment of prices at which the orders are placed (controls) in response to the current inventory and other relevant variables, is required. This gives rise to a natural formulation of MM as a stochastic optimal control problem. Within the most commonly used framework, first used in the seminal Avellaneda-Stoikov (AS) paper [1] and later studied and generalized by a plethora of researchers [2]– [4], the problem boils down to reducing the associated Hamilton-Jacobi-Bellman equation to a system of ordinary differential equations (ODE) and finally deriving closed-form approximations to the optimal MM controls. Alternatively, near-optimal controls could be obtained by techniques such as (deep) reinforcement learning.

This work was supported in part by the Croatian Science Foundation under Project 5241, and in part by the European Regional Development Fund under Grant KK.01.1.1.01.0009 (DATACROSS).

The authors are with the Laboratory for Financial and Risk Analytics, Faculty of Electrical Engineering and Computing, University of Zagreb, 10000 Zagreb, Croatia (e-mail: bruno.gasperov@fer.hr; zvonko.kostanjcar@fer.hr).

Reinforcement learning (RL) is a class of trial-and-error-based methods for sequential decision making under uncertainty, used to solve stochastic optimal control problems framed as Markov decision processes (MDP). RL methods are particularly powerful when combined with deep neural networks (which are used as function approximators), coalescing into a novel field of deep reinforcement learning (DRL). Recent years have witnessed a multitude of striking DRL achievements in a diverse set of domains ranging from the game of Go [5], to text generation [6] and networked control systems [7]. Unsurprisingly, there is nowadays a surge of interest in the applications of (D)RL to the problem of MM, which is naturally framed as a discrete-time MDP.

In [8], the authors present what they claim to be the first practical application of RL to optimal MM in high-frequency trading, using the LOB model proposed in [9] to train an RL agent via simple discrete Q-Learning. The authors then demonstrate that their framework outperforms the AS and fixed-offset benchmarks. Somewhat similarly, Spooner et al. [10] use the SARSA algorithm with a linear combination of tile codings as a value function approximator to produce an RL agent demonstrating superior out-of-sample performance across multiple securities. Sadighian [11] develops a framework for cryptocurrency MM employing two advanced policy gradient-based RL algorithms (A2C and PPO) and a state space comprised of both LOB data and order flow arrival statistics. Gašperov and Kostanjčar [12] present a framework underpinned by ideas from adversarial RL and neuroevolution, with experimental results demonstrating its superior reward-to-risk performance. Other notable approaches introduce additional features like the presence of a dark pool [13], multi-asset MM in corporate bonds [14], and dealer markets [15].

Regardless of the approach used, faithful LOB modeling, ideally accounting for the empirical properties and stylized facts of market microstructure as well as the discrete nature of the LOB itself, is pivotal to obtaining high-performing MM controllers. However, due to the naive assumptions they are predicated upon, the LOB models underlying most contemporary MM approaches remain inconsistent with respect to direction, timing, and volume, leading to phantom gains under backtesting and preposterous events [16], such as price decreases after a large buy market order. For example, in the original AS model [1], price movements are assumed to be completely independent of the arrivals of market orders and the LOB dynamics, while the subsequent approaches only

partly address such inconsistencies. To ameliorate this, a novel weakly-consistent pure-jump market model that ensures that the price dynamics are consistent with the LOB dynamics with respect to direction and timing is proposed in [16]. Nevertheless, it still assumes constant order arrival intensities, meaning that any (empirically found) effects of self- or mutual-excitation and inhibition between various types of LOB order arrivals remain unaccounted for.

In this paper, we consider the stochastic optimal control problem of a market maker trading in a weakly consistent, multivariate Hawkes process-based LOB model. To the authors' knowledge, our approach is the first to wed DRL to MM under such a model. On one hand, our goal is to increase the realism of existing MM models, such as the model proposed in [16]. Given that Hawkes processes are commonly used for realistic modeling of market microstructure [17]–[19], we opt for such a model, which is then used as a backbone of our simulator. On the other hand, we also need to ensure not to jeopardize tractability. To this end, we avoid the use of full LOB models (because of the associated complexities and issues [16]) and choose a reduced-form LOB model instead. Furthermore, a number of additional simplifying assumptions about microstructural dynamics are made. Therefore, underpinning our work is a careful consideration of the balance between complexity and tractability. The model is also interactive in the sense that the market maker's controls (choice of placed limit and market orders) affect the order arrival intensities, i.e., the market responds dynamically to the market maker's control strategy. The proposed approach contributes to the (in our view) underrepresented line of research on MM under weakly consistent LOB models [16]. It also enables the leveraging of the advantages of Monte Carlo backtesting, particularly its ability to perform controlled randomized experiments [20].

II. Preliminaries

A. Multivariate Hawkes processes

A p-dimensional linear Hawkes process [21] is a p-dimensional point process $N(t) = (N_k(t) : k = 1, \dots, p)$ with the intensity of N_k (the k-th dimension) given by:

$$\lambda_k(t) = \mu_k + \sum_{l=1}^{p} \int_0^{t-} f_{k,l}(t-s)\mathrm{d}N_l(s), \tag{1}$$

where $\mu_k \geq 0$ are the baseline intensities, $N_l(t)$ the number of arrivals within $[0, t]$ corresponding to the l-th dimension, and $f_{k,l}(t)$ the kernels (triggering functions). Arrivals in dimension l perturb the intensity of the arrivals in dimension k at time t by $f_{k,l}(t-s)$ for $t > s$. Multivariate linear Hawkes processes are relatively easy to handle and lend themselves well to simulation and interpretation due to their branching structure representation [22]. In cases when exponential kernels are used, the intensities are given by:

$$\lambda_k(t) = \mu_k + \sum_{l=1}^{p} \int_0^{t-} \alpha_{k,l} e^{-\beta_{k,l}(t-s)}\mathrm{d}N_l(s), \tag{2}$$

where $\beta_{k,l} \geq 0$ and $\alpha_{k,l} \geq 0$ are the decay and excitation parameters.

Type of order	Mid-price	Bid-price	Ask-price
1 - Aggressive market buy	+	no effect	+
2 - Aggressive market sell	-	-	no effect
3 - Aggressive limit buy	+	+	no effect
4 - Aggressive limit sell	-	no effect	-
5 - Aggressive limit buy cancelation	-	-	no effect
6 - Aggressive limit sell cancelation	+	no effect	+
7 - Non-aggressive market buy	no effect	no effect	no effect
8 - Non-aggressive market sell	no effect	no effect	no effect
9 - Non-aggressive limit buy	no effect	no effect	no effect
10 - Non-aggressive limit sell	no effect	no effect	no effect
11 - Non-aggressive limit buy cancelation	no effect	no effect	no effect
12 - Non-aggressive limit sell cancelation	no effect	no effect	no effect

Fig. 1. Types of orders and their effect on the mid-price, bid, and ask price.

III. Model

A. Event types

We employ the categorization of LOB events proposed by Biais [23], comprising 12 different types of LOB events, as provided in Fig. 1. Untypical orders, such as iceberg orders or (partly) hidden orders, are omitted from consideration for simplicity's sake. Aggressive market orders, limit orders, and cancellations (events of type 1–6) affect either the bid or the ask price, and consequently the mid-price as well. Non-aggressive market orders (type 7–8) do not affect any of the prices; however, they generate trades and affect the volume at the top of the LOB and are hence relevant to MM. Non-aggressive limit orders and cancellations (type 9–12) affect neither the prices nor the volume at the top of the LOB and are hence ignored. Consequently, we consider 8 event types in total:

$$E_{\text{all}} = \{M_b^a, M_s^a, L_b^a, L_s^a, C_b^a, C_s^a, M_b^n, M_s^n\}, \tag{3}$$

where the superscript denotes the (non)-aggressiveness, and the subscript the side (buy/sell). The resulting LOB model is weakly consistent following [16]. Allowing for variable jump sizes, the LOB dynamics are modeled by an 8-variate marked point process, where marks indicate jump sizes. The corresponding counting process is given by:

$$N(t) = \left(N_{M_b^a}(t), \dots, N_{M_s^n}(t)\right), \tag{4}$$

and the associated intensity vector by:

$$\lambda(t) = \left(\lambda_{M_b^a}(t), \dots, \lambda_{M_s^n}(t)\right). \tag{5}$$

After denoting the jump size (in ticks) associated with an individual event e by J_e, the mid-price P_t is given by:

$$P_t = P_0 + \left(\sum_{e,\ T(e)\in E_{\text{inc}}} J_e \ - \sum_{e,\ T(e)\in E_{\text{dec}}} J_e \right) \frac{\delta}{2}, \tag{6}$$

where P_0 is the initial price, δ the tick size, $T(e)$ the type of event e, $E_{inc} = \{M_b^a, L_b^a, C_s^a\}$, and $E_{dec} = \{M_s^a, L_s^a, C_b^a\}$. Jump sizes are, for the sake of simplicity, assumed to be independent of the jump times and i.i.d. (similarly as in [16]).

B. Simulation procedure

A multivariate linear Hawkes process with exponential kernels is used to model dependencies, namely cross-excitation and self-excitation effects, between different LOB order arrivals (i.e. event types). The choice of exponential kernels is motivated by both their tractability to simulation and suitability for modeling short-term memory processes describing market microstructure, due to which they are used traditionally in financial applications [24] [25]. Furthermore, their use conveniently results in the Markovian properties of the process [17] [26]. In order to simulate the multivariate Hawkes process, we rely on Ogata's modified thinning algorithm [27].

C. Market making procedure

The MM procedure roughly follows the one described in [1]. At the start of each time-step, at time t, the agent (controller) cancels its outstanding limit orders (if there are any), and observes the state of the environment S_t containing market and agent-based features. The agent uses this information to select the action A_t - it decides whether to post limit orders (and at which prices) or a market order. If the absolute value of the agent's inventory is equal to the inventory constraint c, $c \in \mathcal{N}$, the order on the corresponding side is ignored. All relevant market (mid-price, bid and ask price, spread) and agent-based (inventory, cash) variables are then updated accordingly. Next, the LOB events generated by the simulation procedure are processed sequentially, which is followed by corresponding updates of the relevant variables. Executed limit orders cannot be replaced with new ones until the beginning of the next time-step. Finally, the agent reaches the end of the time-step and receives the reward $R_{t+\Delta t}$. When time $t + \Delta t$ is reached, unexecuted limit orders from the previous time-step are canceled, the agent observes the new state of the environment $S_{t+\Delta t}$, selects the action $A_{t+\Delta t}$ and the procedure is iterated until the terminal time T. The agent's inventory I_t is described by the following relation:

$$\mathrm{d}I_t = \mathrm{d}N_t^{\mathrm{b}} - \mathrm{d}N_t^{\mathrm{a}} + \mathrm{d}N_t^{\mathrm{mb}} - \mathrm{d}N_t^{\mathrm{ms}}, \tag{7}$$

where N_t^{b}, N_t^{a}, N_t^{mb}, and N_t^{ms} denote the number of the agent's limit bid (buy), limit ask (sell), market buy, and market sell orders executed up to time t, respectively. The process N_t^{b} is described by:

$$\mathrm{d}N_t^{\mathrm{b}} = \mathrm{d}N_{M_s^a}\mathbb{1}_{\text{fill, } M_s^a} + \mathrm{d}N_{M_s^n}\mathbb{1}_{\text{fill, } M_s^n}, \tag{8}$$

where $\mathbb{1}_{\text{fill, } M_s^a}$ ($\mathbb{1}_{\text{fill, } M_s^n}$) is the indicator function for whether the incoming (non)-aggressive market order fills the market maker's limit order. The process N_t^{a} is described analogously. Finally, the agent's cash process X_t is given by:

$$\mathrm{d}X_t = Q_t^a \mathrm{d}N_t^{\mathrm{a}} - Q_t^b \mathrm{d}N_t^{\mathrm{b}} - (P_t^{\mathrm{a}} + \epsilon_t)\mathrm{d}N_t^{\mathrm{mb}} + (P_t^{\mathrm{b}} - \epsilon_t)\mathrm{d}N_t^{\mathrm{ms}}, \tag{9}$$

where Q_t^a (Q_t^b) denotes the price at which the agent's ask (bid) quote is posted, P_t^{a} (P_t^{b}) the best ask (bid) price, and ϵ_t the additional costs due to fees and market impact, all at time t. Furthermore, a number of simplifying assumptions are made:

- Market orders submitted by the market maker are aggressive with probability Z_1, and its limit order cancellations are aggressive with probability Z_2.
- Exponential distribution with density given by $f(x) = \frac{1}{\beta}\exp\left(-\frac{x-\mu}{\beta}\right)$ is used for modeling the size of the price jumps J_e associated with aggressive events, where μ is the location and β the scale parameter. Where required (i.e. with aggressive limit orders and cancellations), the truncated exponential distribution with the corresponding parameters is used.
- Upon arrival of a non-aggressive market order, the probability of execution of the market maker's limit order standing at the best bid/ask price is fixed and given by Z_3. The limit order is either executed in its entirety or not at all.

IV. REINFORCEMENT LEARNING CONTROLLER

A. State space

The state space consists of the current inventory I_t, the current bid-ask spread Δ_t, and the trend variable α_t:

$$S_t = (I_t, \Delta_t, \alpha_t). \tag{10}$$

Due to the inventory constraints, there are $2c + 1$ possible inventory states, i.e., $I_t \in \{-c, \ldots, c\}$. The current bid-ask spread Δ_t is measured in ticks and strictly positive. The variable α_t accounts for the trend and is calculated as $\alpha_t = \lambda_{M_b^a}(t) + \lambda_{M_b^n}(t) - \lambda_{M_s^a}(t) - \lambda_{M_s^n}(t)$. The inventory feature I_t is normalized by the min-max normalization. Since features Δ_t and α_t have unknown means and variances, we generate a long trajectory with 100,000 steps by a controller that outputs completely random actions and use the obtained means and variances for z-score normalization.

B. Action space

The action (i.e. the controls) A_t at time t corresponds to a pair of offsets from the best bid (ask) prices. Hence:

$$A_t = (Q_t^a - P_t^{\mathrm{a}}, P_t^{\mathrm{b}} - Q_t^b). \tag{11}$$

The agent is allowed to post limit orders at all possible prices, thereby determining the level of aggressiveness/conservativeness on each of the sides of the LOB. If the agent posts a limit order pair with a negative quoted spread, the action is ignored. A bid (ask) order posted at or above (below) the best ask (bid) is treated as a buy (sell) market order and executed immediately. All limit and market orders sent by the market maker are assumed to be of unit size (i.e. are for one unit of the asset) and the controls are rounded to a multiple of the tick size.

C. Reward function

The goal of the MM agent is to maximize the expectation of the terminal wealth while simultaneously minimizing the inventory risk. We take inspiration from the commonly used MM formulation described in [4] and assume that the market maker maximizes the following expression:

$$\mathrm{E}_\pi\left[W_T - \phi\int_0^T |I_t| dt\right], \tag{12}$$

over the set of RL policies. Each policy $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$ maps states to probability distributions over the action space.

$W_t = I_t P_t + X_t$ is the total wealth at time t, T the terminal time, and $\phi \geq 0$ the running inventory penalty parameter which is used to disincentivize the market maker from holding non-zero positions and thereby exposing itself to the inventory risk. Note that, unlike in [4], we use the absolute value of the inventory instead of the quadratic inventory penalty to obtain a convenient value at risk (VaR) interpretation. Therefore, the reward at time $t + \Delta t$ is given by:

$$R_{t+\Delta t} = \Delta W_{t+\Delta t} - \phi \int_t^{t+\Delta t} |I_s| ds. \tag{13}$$

The integrand is piecewise constant and hence trivial.

D. Controller design

A neural network with 2 fully-connected hidden layers of 64 neurons with ReLU activation is used to represent the controller. Such a relatively shallow architecture is somewhat of a standard in DRL and, moreover, simple NN architectures have been shown before to perform comparably to (or better than) sophisticated CNN-LSTM architectures for modeling of the LOB [28].

E. Training

SAC (Soft Actor-Critic) is used to train the RL controller in our experiments. SAC is a state-of-the-art actor-critic RL algorithm capable of tackling continuous action spaces, and characterized by increased robustness and the ability to learn multiple modes of near-optimal behavior. It is a maximum entropy algorithm, meaning that it maximizes not only the return but also the policy entropy, thereby improving exploration. Alternatively, DQN might be used; however, our experimentation showed that it suffers from severe instability issues, especially in its vanilla variant. TD3 (Twin Delayed DDPG) has also been considered but was eventually excluded due to its inferior performance. The number of training time-steps is set to 10^6.

V. EXPERIMENTS

In order to pit the performance of our DRL approach against MM benchmarks, we perform Monte Carlo simulations (back-tests) using synthetic data generated by the simulator. The number of simulations is set to 10^3. Standard MM benchmarks like the AS [1] approximations are ill-suited for our framework since they take into account neither the existence of the bid-ask spread nor the discrete nature of the underlying LOB [29]. We hence need to turn to alternative benchmarks. We consider a class of MM strategies linear in inventory and including inventory constraints. Among the members of this class we denote the best performing one as the LIN strategy. Note that this class of strategies also includes the state-of-the-art Gueant-Lehalle-Fernandez-Tapia (GLFT) [3] approximations. Additionally, we consider a simple strategy (SYM strategy) that always places limit orders precisely at the best bid and the best ask.

A. Risk and performance metrics

Each of the strategies is evaluated by a number of risk/performance metrics, especially including the following:

- **Profit and Loss (PnL) distribution** statistics

TABLE I
DISTRIBUTIONAL STATISTICS — DRL CONTROLLER VS BENCHMARKS

Metric	DRL	SYM	LIN
Mean episode return	13.3378	8.3913	7.6772
Mean PnL	13.7567	9.8686	8.2105
Std PnL	8.4952	8.2443	5.7884
Kurtosis PnL	4.2083	6.0552	10.5585
Skew PnL	1.5870	1.6375	2.3863
Jarque Bera PnL	578.8301	987.3005	2797.0786
Jarque Bera PnL p-value (5%)	0	0	0
10th percentile PnL	4.9445	1.6775	2.7400
20th percentile PnL	7.0480	3.6220	3.9800
80th percentile PnL	19.0340	15.0000	11.4360
90th percentile PnL	24.4310	19.6150	14.7170
Sharpe Ratio	1.6193	1.1970	1.4184
Abs. mean terminal inv.	0.454	1.46	0.56
Mean terminal inv.	-0.122	-0.028	-0.028
Std terminal inv.	0.7477	1.7650	0.8070
Kurtosis inv.	1.6838	-1.0332	0.0277
Skew inv.	0.3742	0.0032	0.1879
Jarque Bera inv.	70.7342	22.2388	2.9569
Jarque Bera inv. p-value (5%)	0	0	0.2280
10th percentile inv.	-1	-2	-1
20th percentile inv.	-1	-2	-1
80th percentile inv.	0	2	1
90th percentile inv.	1	2	1
Mean Absolute Position (MAP)	0.4232	1.4659	0.5478
(Mean PnL)/MAP	32.5064	6.7321	14.9881

- **Mean episode return**
- **Mean Absolute Position (MAP)**, approximated as:

$$\text{MAP} = \frac{1}{N} \sum_{k=1}^{N} |I_{k\Delta t}|, \tag{14}$$

  where N is the number of time-steps in an episode.
- **Sharpe ratio**, which, in the context of high-frequency trading and MM, is commonly defined [30] as:

$$SR = \frac{\mu_{W_T}}{\sigma_{W_T}}, \tag{15}$$

  where μ_{W_T} (σ_{W_T}) denotes the mean (standard deviation) of the terminal wealth (PnL).
- **(Mean PnL)/MAP** - the ratio of the mean terminal wealth (PnL) to the MAP. This is a non-rolling variant of the risk metric introduced in [12].

B. Experimental results

The detailed results (distributional statistics for the PnL and terminal inventory distribution as well as additional risk metrics for all of the strategies) are provided in Table I. The strategies' terminal PnL and inventory distributions are shown in Fig. 2 together with Gaussian kernel density estimates. The results clearly indicate that the DRL strategy outperforms both of the benchmarks with respect to the vast majority of the considered metrics. The DRL strategy performance results, when juxtaposed with the SYM and LIN strategy, show a considerably higher mean PnL value as well as a more favorable Sharpe ratio. The percentiles of the PnL distribution (also interpreted as VaR) indicate its dominant performance as well. As expected, its MAP is fairly low, owing to the effect of inventory penalization, implying that the strategy succeeds at MM without exposing the trader to high inventory risks. Also note the significantly lower kurtosis value for its PnL distribution, indicating thinner tails, which is a desirable property for risk management purposes, as well as a somewhat smaller PnL distribution skewness. We also

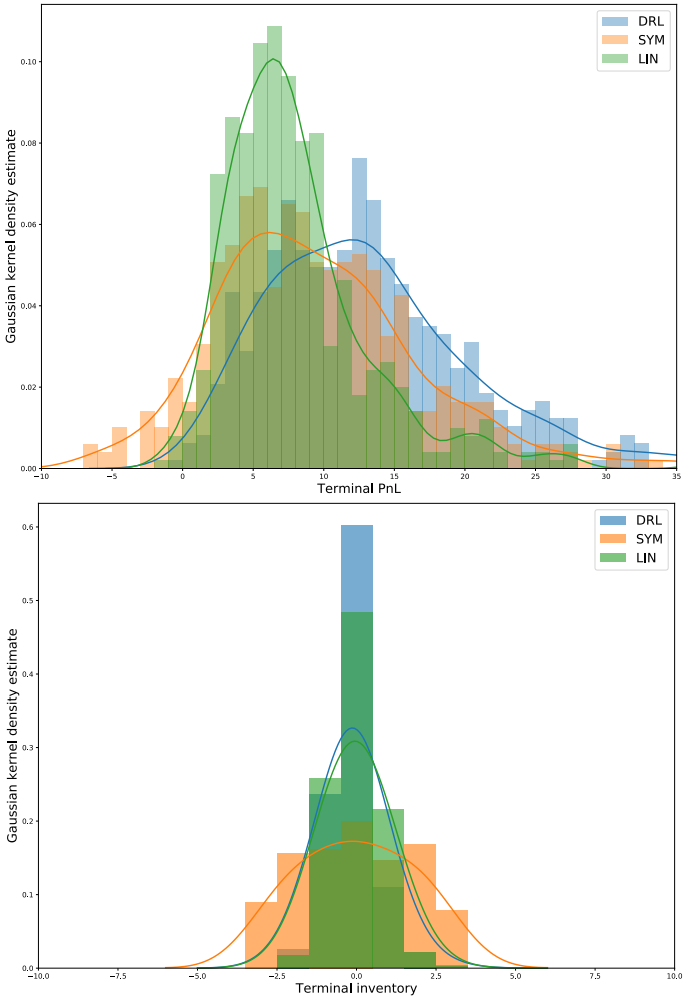


Fig. 2. PnL and terminal inventory distributions — DRL versus benchmarks

observe that the Jarque-Bera test results indicate inconsistency with the null hypothesis (normality of the PnL distribution) for all of the strategies. Interestingly, comparing the SYM and the LIN strategy, the latter exhibits lower risk at the expense of lower expected PnL, similarly to the results from the AS study [1]. The terminal inventory distribution for all of the strategies is centered around zero, demonstrating telltale signs of MM behavior (position oscillating around zero inventory). We observe that the terminal inventory distributions corresponding to the DRL and the LIN strategy seem quite similar, indicating comparable risk preferences. On the contrary, notice the size of the absolute mean terminal inventory distribution corresponding to the SYM strategy, clearly indicating its naive disregard for the inventory risk. Fig. 3 depicts the mean performance of the DRL strategy in time together with the associated confidence intervals, with the mean PnL growing (seemingly) linearly in time.

C. Sensitivity analysis

Ideally, we would like to obtain MM controls robust to changes in the underlying order arrival intensity rates. Relatedly, we conduct sensitivity analysis of the DRL controller by changing the background intensity rates for all order types — specifically by adding normal (Gaussian) noise.

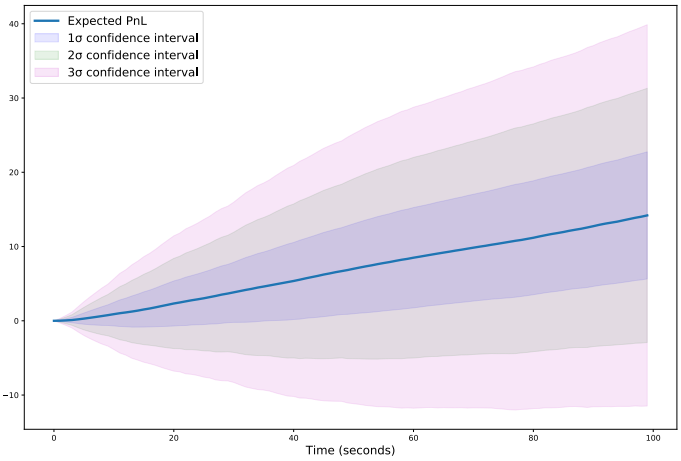


Fig. 3. DRL strategy — mean PnL in time

More precisely, three different noise sizes were considered — Gaussian noise based with mean 0 and variance 0.1 (DRL-N-0.1), variance 0.2 (DRL-N-0.2) and variance 0.3 (DRL-N-0.3). The results are shown in Table II. Evidently, large perturbations increase the standard deviation of the terminal PnL distribution, while the effect on the mean is more complicated, with the largest noise size resulting in the largest mean PnL (and also MAP). Quite expectedly, the Sharpe ratio clearly decreases (while the MAP increases) as the size of the perturbation grows.

TABLE II
DISTRIBUTIONAL STATISTICS — RL NOISE LEVELS

Metric	DRL-N-0.1	DRL-N-0.2	DRL-N-0.3
Mean PnL	13.2050	13.2553	15.5077
Std PnL	9.5922	13.0264	17.9523
Kurtosis PnL	4.5894	15.2773	12.0886
Skew PnL	1.6481	3.0457	2.9768
Jarque Bera PnL	665.14	5635.43	3782.90
Jarque Bera PnL p-value	0	0	0
10th percentile PnL	3.7090	2.3745	1.9472
20th percentile PnL	5.4676	4.4940	3.6387
80th percentile PnL	19.0860	19.7860	22.2980
90th percentile PnL	24.7940	26.3665	34.6055
Sharpe Ratio	1.3766	1.0175	0.8638
Mean Absolute Position	0.4443	0.5038	0.5486

Furthermore, we investigate the performance of our strategy and the benchmark strategies under varying transaction costs. Fig. 4 shows the Sharpe ratio for the three strategies and variable limit order transaction fee rates. It is clear from the figure that the DRL strategy is capable of tolerating the inclusion of fees for limit order transactions, up to the fee of around 0.6%. On the contrary, the LIN strategy generates negative profits already with fees set to 0.4%, while the SYM strategy demonstrates extremely low tolerance to such transaction fees, even for the relatively low rate of 0.2%. The considered rates lie in a realistic range, and therefore this validates the performance of our DRL method under real-life conditions. Finally, we retrain the DRL controller under varying limit order transaction fees and investigate the corresponding mean number of transactions. The mean number of transactions equals 24.54, 15.0, 11.28 and 7.85 under limit order transaction costs of 0%, 0.2%, 0.4%, and 0.6%, respectively. The results clearly indicate that higher limit order transaction costs, quite expectedly, lead to fewer

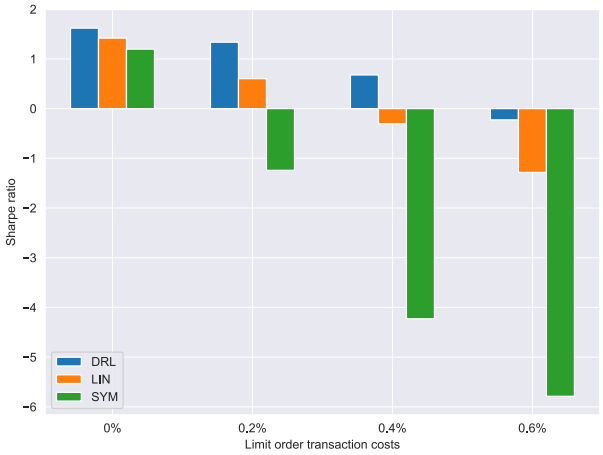


Fig. 4. Sharpe Ratio under variable limit order transaction fees

realized transactions (trades), as the DRL controller becomes increasingly discriminating about when and if to post limit orders in the presence of high transaction costs.

VI. CONCLUSION

A DRL-based approach was used to obtain market-making strategies with superior performance, as compared to heuristic benchmarks. The approach yields promising results when realistic LOB simulators based on multivariate Hawkes processes are employed. Special focus was placed on the statistical analysis of the resulting PnL and terminal inventory distributions, and sensitivity analysis, both to changes in the underlying order intensity rates and the limit order fees. We conclude that DRL presents a viable approach for obtaining competitive MM controllers. Further research might consider more advanced MM models which take into account the full limit order book, or are based on Hawkes processes with alternative (power-law or polynomial) kernels. Finally, robust adversarial RL [31] might be used to further improve generalization and robustness under model uncertainty.

APPENDIX

The following hyperparameter values were used for training: $\gamma = 1$; batch size, 512; buffer size, 10^5; learning rate, 0.0003; learning starts, 100; ent. coef., "auto"; target update interval, 1; gradient steps, 1; train freq., 1; $\tau = 0.005$. The following MM procedure parameter values were used: $\Delta t = 1$; $T = 100$; $\delta = 0.01$; $\phi = 0.01$; $c = 3$; $\mu = 0.01$; $Z_1 = \frac{8}{30}$; $Z_2 = 0.25$; $Z_3 = 0.25$; $\beta = 0.08$; market maker fee, 0%; market taker fee, 0.2%. The Stable Baselines package was used for the implementation. The code is provided at: `github.com/BGasperov/drlformm`

ACKNOWLEDGMENTS

We would like to thank the three anonymous reviewers and the editor for their extremely valuable and helpful comments.

REFERENCES

[1] Avellaneda, M. and Stoikov, S., 2008. High-frequency trading in a limit order book. *Quantitative Finance, 8*(3), pp.217-224.

[2] Fodra, P. and Pham, H., 2015. High frequency trading and asymptotics for small risk aversion in a Markov renewal model. *SIAM Journal on Financial Mathematics, 6*(1), pp.656-684.

[3] Guéant, O., Lehalle, C.A. and Fernandez-Tapia, J., 2013. Dealing with the inventory risk: a solution to the market making problem. *Mathematics and financial economics, 7*(4), pp.477-507.

[4] Cartea, A., Jaimungal, S. and Ricci, J., 2018. Algorithmic trading, stochastic control, and mutually exciting processes. *SIAM Review, 60*(3), pp.673-703.

[5] Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M. and Dieleman, S., 2016. Mastering the game of Go with deep neural networks and tree search. *Nature, 529*(7587), pp.484-489.

[6] Ranzato, M.A., Chopra, S., Auli, M. and Zaremba, W., 2015. Sequence level training with recurrent neural networks. *arXiv preprint arXiv:1511.06732*.

[7] Demirel, B., Ramaswamy, A., Quevedo, D.E. and Karl, H., 2018. Deepcas: A deep reinforcement learning algorithm for control-aware scheduling. *IEEE Control Systems Letters, 2*(4), pp.737-742.

[8] Lim, Y.S. and Gorse, D., 2018, April. Reinforcement learning for high-frequency market making. In *ESANN 2018-Proceedings, European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning* (pp. 521-526). ESANN.

[9] Cont, R., Stoikov, S. and Talreja, R., 2010. A stochastic model for order book dynamics. *Operations research, 58(3)*, pp.549-563.

[10] Spooner, T., Fearnley, J., Savani, R. and Koukorinis, A., 2018. Market making via reinforcement learning. *arXiv preprint arXiv:1804.04216*.

[11] Sadighian, J., 2019. Deep reinforcement learning in cryptocurrency market making. *arXiv preprint arXiv:1911.08647*.

[12] Gašperov, B. and Kostanjčar, Z., 2021. Market Making With Signals Through Deep Reinforcement Learning. *IEEE Access, 9*, pp.61611-61622.

[13] Baldacci, B., Manziuk, I., Mastrolia, T. and Rosenbaum, M., 2019. Market making and incentives design in the presence of a dark pool: a deep reinforcement learning approach. *arXiv preprint arXiv:1912.01129*.

[14] Guéant, O. and Manziuk, I., 2019. Deep reinforcement learning for market making in corporate bonds: beating the curse of dimensionality. *Applied Mathematical Finance, 26*(5), pp.387-452.

[15] Ganesh, S., Vadori, N., Xu, M., Zheng, H., Reddy, P. and Veloso, M., 2019. Reinforcement learning for market making in a multi-agent dealer market. *arXiv preprint arXiv:1911.05892*.

[16] Law, B. and Viens, F., 2019. Market making under a weakly consistent limit order book model. *High Frequency, 2*(3-4), pp.215-238.

[17] Bacry, E., Mastromatteo, I. and Muzy, J.F., 2015. Hawkes processes in finance. *Market Microstructure and Liquidity, 1*(01), p.1550005.

[18] Lu, X. and Abergel, F., 2018. High-dimensional Hawkes processes for limit order books: modelling, empirical analysis and numerical calibration. *Quantitative Finance, 18*(2), pp.249-264.

[19] Da Fonseca, J. and Zaatour, R., 2014. Hawkes process: Fast calibration, application to trade clustering, and diffusive limit. *Journal of Futures Markets, 34*(6), pp.548-579.

[20] Lopez de Prado, M., 2019. Tactical investment algorithms. *Available at SSRN 3459866*.

[21] Embrechts, P., Liniger, T. and Lin, L., 2011. Multivariate Hawkes processes: an application to financial data. *Journal of Applied Probability, 48*(A), pp.367-378.

[22] Law, B. and Viens, F., 2016. Hawkes processes and their applications to high-frequency data modeling. *Handbook of High-Frequency Trading and Modeling in Finance, 9*, p.183.

[23] Biais, B., Hillion, P. and Spatt, C., 1995. An empirical analysis of the limit order book and the order flow in the Paris Bourse. *the Journal of Finance, 50*(5), pp.1655-1689.

[24] Filimonov, V. and Sornette, D., 2015. Apparent criticality and calibration issues in the Hawkes self-excited point process model: application to high-frequency financial data. *Quantitative Finance, 15*(8), pp.1293-1314.

[25] Simon, G., 2016. *Hawkes Processes in Finance: A Review with Simulations* (Doctoral dissertation, University of Oregon).

[26] Oakes, D., 1975. The Markovian self-exciting process. *Journal of Applied Probability, 12*(1), pp.69-77.

[27] Ogata, Y., 1981. On Lewis' simulation method for point processes. *IEEE Transactions on Information Theory, 27*(1), pp.23-31.

[28] Briola, A., Turiel, J. and Aste, T., 2020. Deep Learning modeling of Limit Order Book: a comparative perspective. *arXiv preprint arXiv:2007.07319*.

[29] Guéant, O., 2016. *The Financial Mathematics of Market Liquidity: From optimal execution to market making* (Vol. 33). CRC Press.

[30] Coates, J.M. and Page, L., 2009. A note on trader Sharpe Ratios. *PloS one, 4*(11), p.e8036.

[31] Pinto, L., Davidson, J., Sukthankar, R. and Gupta, A., 2017, July. Robust adversarial reinforcement learning. In *International Conference on Machine Learning* (pp. 2817-2826). PMLR.