Deep Learning with Reinforcement Learning on Order Books

Koti S. Jaddu and Paul A. Bilokon

Koti S. Jaddu
is an MSc computing (artificial intelligence and machine learning) graduate at Imperial College London in London, UK.
koti.jaddu22@imperial.ac.uk

Paul A. Bilokon
is the CEO and founder of Thalesians in Canary Wharf, London, and a visiting professor at Imperial College London in London, UK.
paul.bilokon@imperial.ac.uk

KEY FINDINGS

- High-frequency trading relies on quick automated decisions to exploit price imbalances, but current analytical methods often focus on limited market domains.
- Combining deep learning on order books with reinforcement learning provides a more manageable and lightweight approach to large-scale end-to-end learning.
- The authors concentrate on forecasting returns across various time horizons using order flow imbalance, and three temporal-difference learning models are trained on five different financial instruments.

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. Although 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 ever-growing 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 article 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. Through backtesting and forward testing, the results prove potential but require further minimal modifications for consistently profitable trading to fully handle retail trading costs, slippage, and spread fluctuation.

In 1992, an Internet revolution disrupted the financial trading industry when the first online brokerage service provider, E*Trade, was launched (Wu, Siegel, and Manion 1999, p. 3). 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 (Breckenfelder 2020, p. 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 in less than 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 born, and as hardware improved along with an abundance of data, so did its potential. Lahmiri and Bekiros (2021) showcase deep learning accurately forecasting Bitcoin's high-frequency price data. Kolm, Turiel, and Westray (2021) 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 traders could input their order. Reinforcement learning (RL) 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 (2021) trained a few profitable deep RL agents using long, mid-, and short-term mean-reverting signals as features. The following article combines Bertermann's RL with Kolm, Turiel, and Westray's alpha extraction and investigates its potential in a realistic retail trading setting.

The "Related Work" section describes the relevant literature to showcase existing ideas. This describes different order book feature extraction methods, recent works on using supervised learning and RL on the order books, and finally a list of popular performance metrics used to evaluate trading agents.

The "Design, Implementation, and Testing" section discusses the design and implementation of the solutions, which first describes the data collection pipeline. Next, the design of the supervised learning and RL 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.

The "Evaluation" section evaluates all 15 models. The performance of the supervised learning and RL 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 behaviors.

The "Conclusions and Further Work" section 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.

RELATED WORK

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

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 (Köppen 2000), 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 because 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 limit order book (LOB) that 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 10 nonempty 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:

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

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 midprice jumps will be explained: price, volume, and order flow (OF).

Price extraction methods. From initial speculation, removing the volume elements from Equation 1 leaves the price components, and so a valid price extraction method could be the following:

$$\mathbf{s}_t^{price} := (a_t^1, b_t^1, \ldots, a_t^{10}, b_t^{10})^T \in \mathbb{R}^{20} \quad (2)$$

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 midprice, which infers the state of the best bid and ask prices. The calculation for midprice, the most abstracted state of the LOB, is the following:

$$\mathbf{s}_t^{mid-price} := \frac{a_t^1 + b_t^1}{2} \in \mathbb{R} \quad (3)$$

This reduces the number of dimensions representing the state of the market from 40 if using 10 nonempty 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 midprice. Two popular examples include the following:

- Mean reversion strategies identify deviations from the mean calculated across a period of historical midprices and trade hoping price will correct itself toward the mean if deviated above a threshold (CMC Markets 2023a)

- Momentum strategies identify high volatile movements in midprice and trade hoping for price to continue in that direction (CMC Markets 2023b)

Such algorithms require recent historical midprice data, so here is a more appropriate feature extraction method that captures the change in midprice across 10 consecutive time steps:

$$\mathbf{s}_t^{\Delta mid-price} := \left(\frac{(a_i^1 + b_i^1)}{2} - \frac{(a_{i-1}^1 + b_{i-1}^1)}{2} | i \in [t-9, t] \right) \in \mathbb{R}^{10} \tag{4}$$

Volume extraction methods. Similar to the price extraction methods part, the price components from the raw LOB states shown in Equation 1 can be removed, which leaves the aggregated volume values at each nonempty bid and ask level. This would lead to the following extracted feature:

$$\mathbf{s}_t^{volume} := (v_t^{1,a}, v_t^{1,b}, \ldots, v_t^{10,a}, v_t^{10,b})^T \in \mathbb{R}^{20} \tag{5}$$

As mentioned, there is room for improvement. If we sum up the quantities in the ask and bid side separately for the first 10 nonempty price levels, we get the following:

$$\mathbf{s}_t^{\Sigma\, volume} := \left(\sum_{n=1}^{10} v_t^{n,a}, \sum_{n=1}^{10} v_t^{n,b} \right)^T \in \mathbb{R}^2 \tag{6}$$

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):

$$\mathbf{s}_t^{VD} := \sum_{n=1}^{10} (v_t^{n,a} - v_t^{n,b}) \in \mathbb{R} \tag{7}$$

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 time steps. This reveals the change in Equation 7 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:

$$\mathbf{s}_t^{CVD} := \left(\sum_{n=1}^{10} (v_i^{n,a} - v_i^{n,b}) - \sum_{n=1}^{10} (v_{i-1}^{n,a} - v_{i-1}^{n,b}) | i \in [t-9, t] \right)^T \in \mathbb{R}^{10} \tag{8}$$

There are many algorithms that use cumulative volume deltas; notable examples consist of volume delta reversal strategies, which identify a change of sign in the cumulative volume delta and trade expecting price to reverse direction (Kohout 2022).

OF extraction methods. OF 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 OF from the LOB as described in Kolm, Turiel, and Westray (2021).

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

$$\mathrm{aOF}_{t,i} := \begin{cases} v_t^{i,a}, & \text{if } a_t^i < a_{t-1}^i \\ v_t^{i,a} - v_{t-1}^{i,a}, & \text{if } a_t^i = a_{t-1}^i \\ -v_t^{i,a}, & \text{if } a_t^i > a_{t-1}^i \end{cases} \tag{9}$$

$$\mathrm{bOF}_{t,i} := \begin{cases} v_t^{i,b}, & \text{if } b_t^i > b_{t-1}^i \\ v_t^{i,b} - v_{t-1}^{i,b}, & \text{if } b_t^i = b_{t-1}^i \\ -v_t^{i,b}, & \text{if } b_t^i < b_{t-1}^i \end{cases} \tag{10}$$

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

$$\mathrm{OF}_t := \begin{pmatrix} \mathrm{bOF}_t \\ \mathrm{aOF}_t \end{pmatrix} \in \mathbb{R}^{20} \tag{11}$$

Order flow imbalance (OFI) takes this further as it reveals the disparity between the ask and bid OF components. It can be derived as the following:

$$\mathrm{OFI}_t := \mathrm{bOF}_t - \mathrm{aOF}_t \in \mathbb{R}^{10} \tag{12}$$

Supervised Learning on Order Books

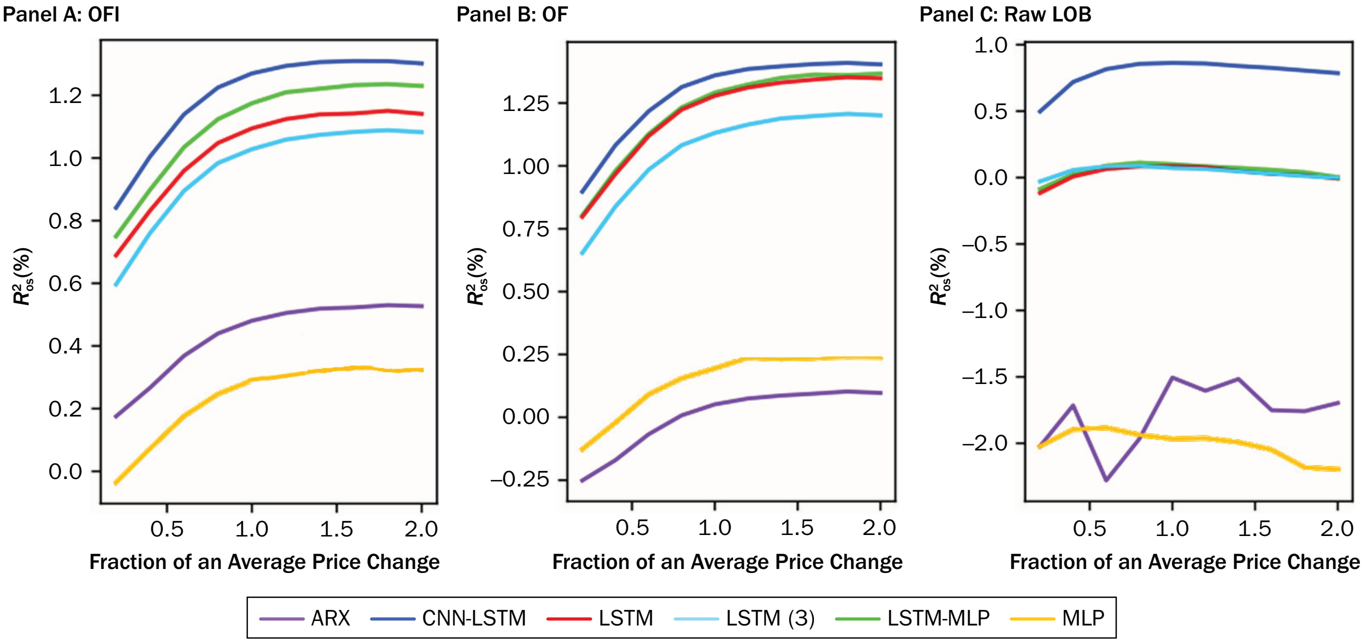
This part describes the recent work on the topic of forecasting small price changes (alpha) using supervised learning on features extracted from the LOB. Recent literature (Mäkinen et al. 2019; Tran et al. 2019; Passalis et al. 2020) independently but commonly used classifiers to predict price movement, either labeling 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 modeling 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 are more data. Kolm, Turiel, and Westray (2021) address this by changing the problem to regression—a better choice because it removes unwanted assumptions but is more computationally demanding.

Kolm, Turiel, and Westray's design extracts alpha at multiple time steps (*horizons*)—alpha is the expected change in price, usually after a very short period. Inspired by Cont, Kukanov, and Stoikov (2014) on stationary quantities derived from the LOB (OF), networks trained on this outperformed networks that were trained from the raw LOB. Kolm, Turiel, and Westray took this idea and found that using OFI 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 limit order book system—the efficient reconstructor (LOBSTER) (Huang and Polak 2011).

Exhibit 1 presents the results of the forecasting performance of selective models against the ratio of horizon period over price change (Kolm, Turiel, and Westray 2021).

EXHIBIT 1

Forecasting Performance of Different Models



Panel A: OFI

Panel B: OF

Panel C: Raw LOB

NOTES: Each panel shows the average out-of-sample R^2 against horizons represented as the fraction of an average price change per stock.

SOURCE: Kolm, Turiel, and Westray (2021).

Each model was trained while adopting a three-week rolling-window out-of-sample methodology across 48 weeks using a (one week validation, four weeks training, one week out-of-sample testing) structure. The evaluation metric used is out-of-sample R^2 (R^2_{OS}). It is defined in Kolm, Turiel, and Westray (2021) as

$$R^2_{\mathrm{OS},h} := 1 - \mathrm{MSE}_{\mathrm{m},h}/\mathrm{MSE}_{\mathrm{bmk},h} \tag{13}$$

for each *horizon h*, where $\mathrm{MSE}_{\mathrm{m},h}$ and $\mathrm{MSE}_{\mathrm{bmk},h}$ 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 Exhibit 1. Overall, the results show that models perform better when using OF and OFI as input rather than raw LOB and that convolutional neural network (CNN)-long short-term memory (LSTM) extracts alpha the most accurately, followed by LSTM-MLP, LSTM, and a stacked LSTM with three layers. MLP stands for multilayer perceptron, which is a stack of linear layers. The R^2_{OS} (%) is mostly positive, meaning that these networks beat the benchmark.

RL on Order Books

The related work done using RL on the LOB to execute or simulate trades will be covered. The first large-scale experiments were conducted in 2006 by Nevmyvaka, Feng, and Kearns 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 evaluated temporal-difference algorithms (including Q-learning) in realistic simulations and observed promising performance. To address the trade-off between maximizing profits and managing risk in trading, Mani, Phelps, and Parsons in 2019 incorporated risk-sensitive RL. This agent's decisions are not only led by potential 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 in which explainability is essential. Vyetrenko and Xu in 2019 addressed this for risk-sensitive RL strategies by representing the agent's decisions in compact decision trees.

Karpe et al. in 2020 used 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 ByBitHelp (2023), 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 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. Exhibit 2 displays the results graphically inferred from the original tabular version shown in Exhibit A1 in the appendix. 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 2,500 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 suboptimal in this study.

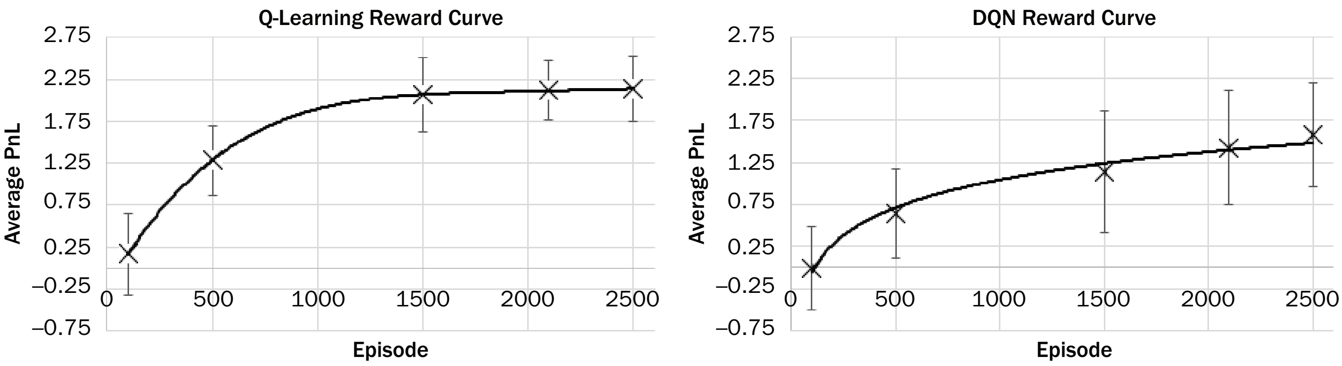
Evaluating Trading Agents

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 Auquan (2017) and Singh and Pachanekar (2019):

- 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

EXHIBIT 2

Graphical Performance Comparison (average PnL) of Q-Learning and DQN



NOTE: This comparison was inferred from a tabular version provided by Bertermann (2021); the error bars are standard deviations.

- 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

The Sharpe and Sortino ratios will be omitted because 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.

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 RL for profitable trading. Two successful studies were selected (one supervised learning and one RL) 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 Work" section, the approach taken forward was adopting Kolm, Turiel, and Westray's (2021) 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 (2021) to cover the "reinforcement learning for profitable trading." Kolm, Turiel, and Westray's alpha extraction needs Bertermann's RL to productize their work, and likewise, Bertermann's RL requires quality features compared to lagging mean-reverting signals (despite observing remarkable results).

Data Collection

This part 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 (Huang and Polak 2011). Although this would also fit well into this work, we wish to integrate the models into our personal retail trading platform to execute automated trades for realistic simulated testing (CTrader FXPro 2023). 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 maximize accuracy.

CTrader FXPro gives users access to the current state of the LOB but does not provide historical data. Hence, LOB data were collected as early as possible (May 23, 2023), saving each LOB state as well as the midprice per time step to a CSV file for easy access. Time steps are defined as changes to the observable LOB states. Storing raw LOB data could bring ethical issues, so the code on CTrader extracts the OFI features, which is what will be used as input to the networks, and stores that into the CSV file instead as in Exhibit 3. The first 10 OFI levels were used by Kolm, Turiel, and Westray (2021) and should be sufficient to carry out this work.

As this data extraction should run continuously for 10 weeks, this was set up on a virtual private server (VPS) hosted by TradingFX VPS, and the files were transferred to local storage every weekend to mitigate the loss upon any technical issue (TradingFX VPS 2023). 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 indexes, foreign exchange pairs, and metal.

Supervised Learning

This part discusses the decisions made during the integration of Kolm, Turiel, and Westray's (2021) alpha extraction into this work. Overall, OFI features will be collected, which will be used as input to a supervised learning regression model that will predict the change in midprice (alpha) for the next six horizons. The reason for choosing six is that Kolm, Turiel, and Westray 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 RL part.

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 will be used, which has more functionality than needed for this work (PyTorch 2023). With regard 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 compute unified device architecture (CUDA) during training to save time (Nvidia 2023).

Pipeline. Algorithm 1 will be the pipeline that includes early stopping–an enhancement from Kolm, Turiel, and Westray (2021). A recommendation from the authors that worked well was to adopt L2 regularization 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 an 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, Turiel, and Westray show that an MLP also produces promising results, so this will be the backbone architecture of the regression networks. Some hyperparameters are set from existing literature as shown in Exhibit 4.

There are many ways to optimize the remainder of the hyperparameters, but because 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. Exhibit 5 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.

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).

Regarding the preprocessing on line 1 in Algorithm 1, limiting the data to remove periods of low trading participation (pre- and postmarket) 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. (2020) for their work, so nothing was removed.

EXHIBIT 3

First Five Records of the CSV File Containing Time, OFI, and Midprice Data

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

Algorithm 1 Supervised Learning Algorithm

```
Apply preprocessing to data
Split data ratio (8 : 1 : 1) into training $T$, validation $V$, and testing $H$ sets
Initialize hyperparameters
Initialize model $M$ and optimizer $Opt$ ($Method, Learning Rate$)
$k \leftarrow 0$ # 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
```

RL

This part discusses the decisions made during the integration of Bertermann's (2021) RL agents (Q-learning and deep Q-learning) into this work. A DDQN will also be investigated as it showed successful results according to Karpe et al. and addresses the problems with the DQN (Karpe et al. 2020).

The code will be written in the Python programming language using PyTorch wherever applicable, as mentioned previously. 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. There are many hyperparameters that will need to be tuned, which will be discussed in the next section.

EXHIBIT 4

Supervised Learning Hyperparameter List from Existing Literature

Hyperparameter	Value
Activation Function	ReLU
Loss Function	MSE
Optimizer Method	ADAM
Maximum Number of Epochs	100
Number of Hidden Layers	4

The agent will 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 (2021) setup.

EXHIBIT 5

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

EXHIBIT 6

RL Hyperparameter List from Existing Literature

Hyperparameter	Value
(All) Maximum Exploration Rate	1.0
(All) Minimum Exploration Rate	0.1
(All) Exploration Rate Decay	0.935
(All) Number of Episodes	80
(All) Reward Function	PNL change
(Q) Number of Buckets per State	5
(DQN, DDQN) Experience Replay Buffer Size	10,000
(DQN, DDQN) Number of Hidden Layers	2
(DQN, DDQN) Activation Function	Sigmoid for Output Layer; ReLU Otherwise
(DQN, DDQN) Optimizer Method	ADAM

EXHIBIT 7

RL 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

From initial experimentation, using exploration decay was found to be important, which decreases the amount of exploration done by the agent over time while emphasizing high exploration during early stages of training. This was recommended by Bertermann. On the other hand, a decay in learning rate did not show any improvements in learning and, in most cases, worsened performance. L2 regularization was also incorporated in the DQN and DDQN models to prevent overfitting by encouraging the weights to be small. Some hyperparameters are set from existing literature as shown in Exhibit 6.

Grid search was used to tune the remainder of the hyperparameters. Exhibit 7 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.

There are 4 combinations for Q-learning, 32 combinations for DQN, and only 2 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).

EVALUATION

This section covers the results of the optimized 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 platform (CTrader FXPro 2023). This section concludes with an investigation to try to explain what values cause the agents to reverse their trades.

Tuning Results

The results from the hyperparameter tuning for the supervised learning and RL components will be presented. 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. Note that the supervised learning and the RL components are separate in this section but will be combined for backtesting on the same test dataset in the next section.

Supervised learning. The supervised learning tuning methodology (Exhibit 8) reveals the optimal parameters for each instrument. 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 10 times larger for XAUUSD, GBPUSD, and EURUSD compared to the indexes FTSE100 and DE40. This suggests that XAUUSD, GBPUSD, and EURUSD are noisier compared to the indexes, which is supported by the high number of data points for the three instruments.

Exhibit 9 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 (leftmost panel). The other models also show overfitting but at a much smaller scale and may be acceptable.

Exhibit 10 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

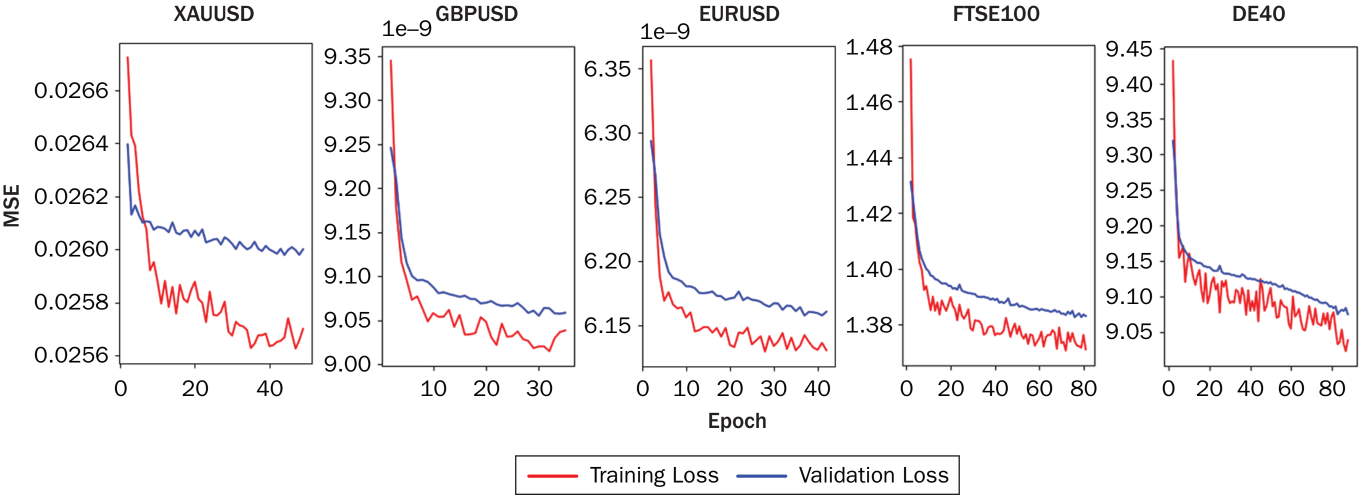
EXHIBIT 8

Best Supervised Learning Hyperparameters from Tuning for Each Instrument

Parameter	XAUUSD, GBPUSD, EURUSD	FTSE100, DE40
Layer/Nodes	[2,048] * 4	[2,048] * 4
Learning Rate	0.0001	0.00001
Early Stopping Parameter	5	5
Batch Size	256	256

EXHIBIT 9

Learning Curves of Alpha Extraction



NOTE: This exhibit shows training MSE loss (red) and validation loss (blue) for each instrument.

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, Turiel, and Westray (2021) as our baseline, which requires the breaking down of the model's performance at the horizon level. This is done in Exhibit 11, and this presents the root mean square error (RMSE) on the test set, the standard deviation of the test set, and the out-of-sample R^2 metric (calculated in Equation 13), which is used by Kolm, Turiel, and Westray to evaluate their work. There is a theme that the RMSE hovers just under the standard deviation, which is alarming because 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, Turiel, and Westray for alpha extraction using an MLP network. They observe an average of 0.181%, whereas the results in Exhibit 11 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.

EXHIBIT 10

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

Kolm, Turiel, and Westray also found that as the fraction of average price change increases (proportional to the horizon), the better the R^2_{OS} (Exhibit 1), and Exhibit 11 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

EXHIBIT 11

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**

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 indexes are outperforming the rest of the instruments, and Kolm, Turiel, and Westray trained their models on stocks from the NASDAQ index, so equities might be ideal for this setup. Although these results agree with Kolm, Turiel, and Westray'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 RL component will give more insight into the true performance of these supervised learning models.

RL. The RL tuning methodology (Exhibits 12 and 13) shows the optimal parameters for each instrument for each agent. All three agents will be evaluated and compared together.

Once again, it is reassuring to see similar parameter values for each instrument, but there were not many options as shown in Exhibit 7. 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 stabilize 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.

EXHIBIT 12

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

EXHIBIT 13

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

EXHIBIT 14

Reward Curves of Q-Learning for Each Instrument during Training (one lot size trades)

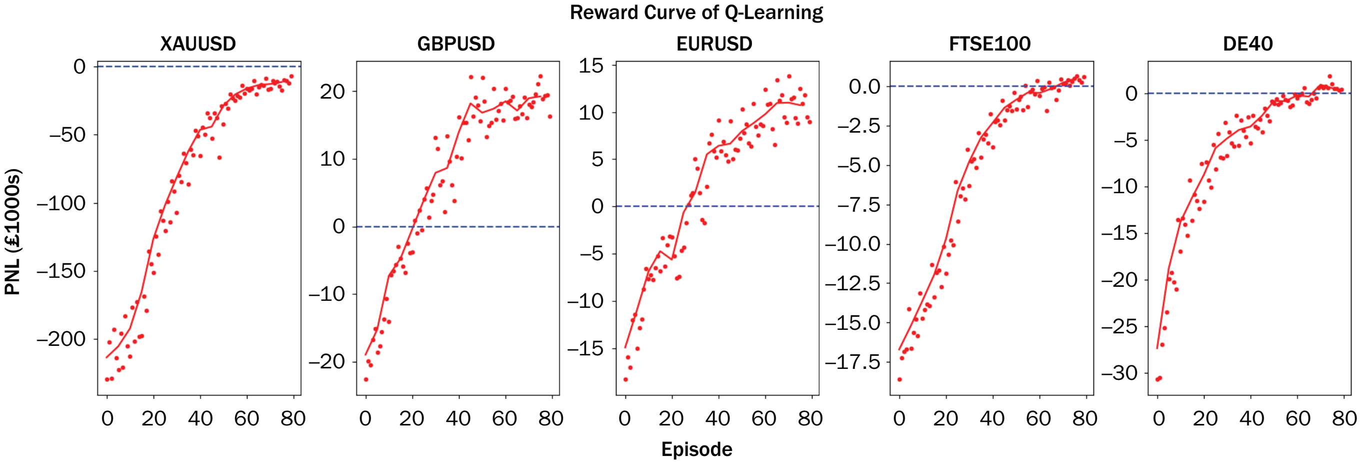


EXHIBIT 15

Reward Curves of DQN for Each Instrument during Training (one lot size trades)

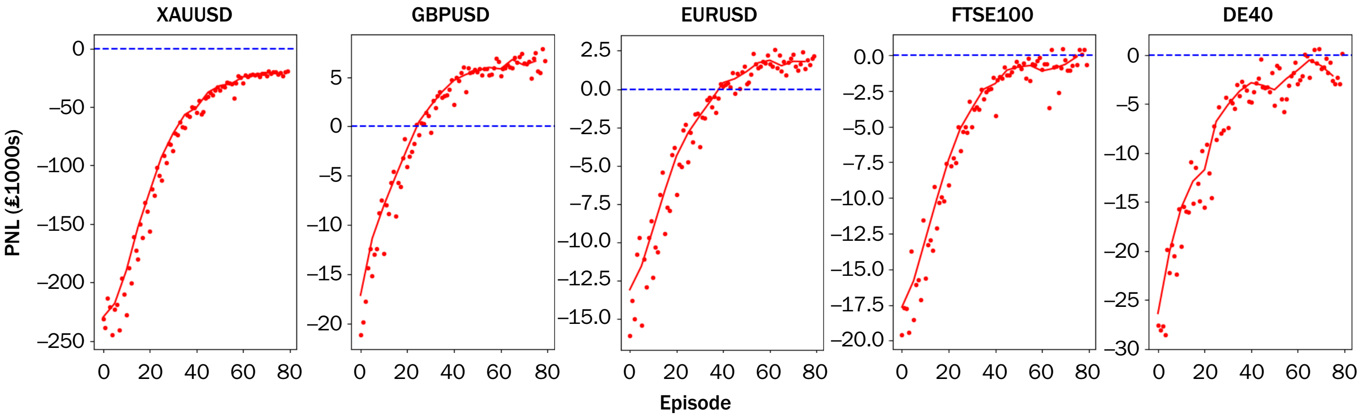
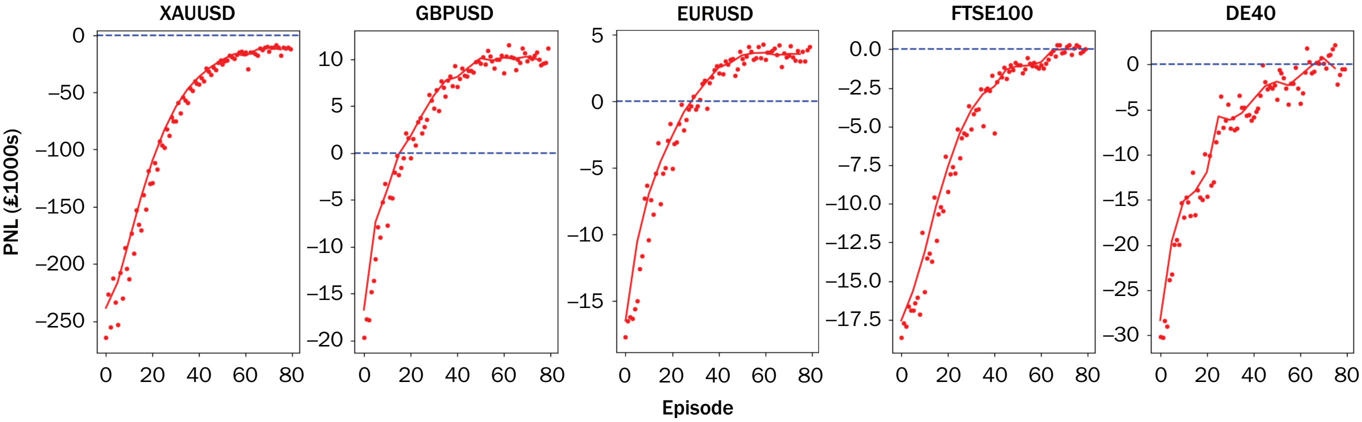


EXHIBIT 16

Reward Curves of DDQN for Each Instrument during Training (one lot size trades)



Exhibits 14, 15, and 16 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 10 weeks. With regard 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 because this is only during training.

Exhibits 17, 18, and 19 show the calculated performance metrics of each agent on the five-day test dataset for each instrument. Overall, the results align with the outcome of the reward curves during training, which indicates that these agents generalized 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.

EXHIBIT 17

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

EXHIBIT 18

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

EXHIBIT 19

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

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 (2021) 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 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.

Backtesting Results

This part combines both the alpha extraction (supervised learning component) and the RL 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. Exhibits 20, 21, and 22 showcase the results of the pipeline on the same test data for comparison to Exhibits 17, 18, and 19.

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 indexes—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 second 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.

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

EXHIBIT 20

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

EXHIBIT 21

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

EXHIBIT 22

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

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 the following), 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 Exhibit 23.

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 five data points (five days of testing), it is better to use a nonparametric test such as the Mann–Whitney *U*-test (Billiet 2003). 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 model needs to be calculated and compared to the critical value in order to reject the null hypothesis.

The steps in Billiet (2003) 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. Because 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.

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. Exhibit 24 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. First, the sample size is too small, and it could be profitable over longer periods of time, so more data are required to make a better judgement on its performance. This is less likely, though. Second and more importantly, the trading bot often skips a few time steps 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

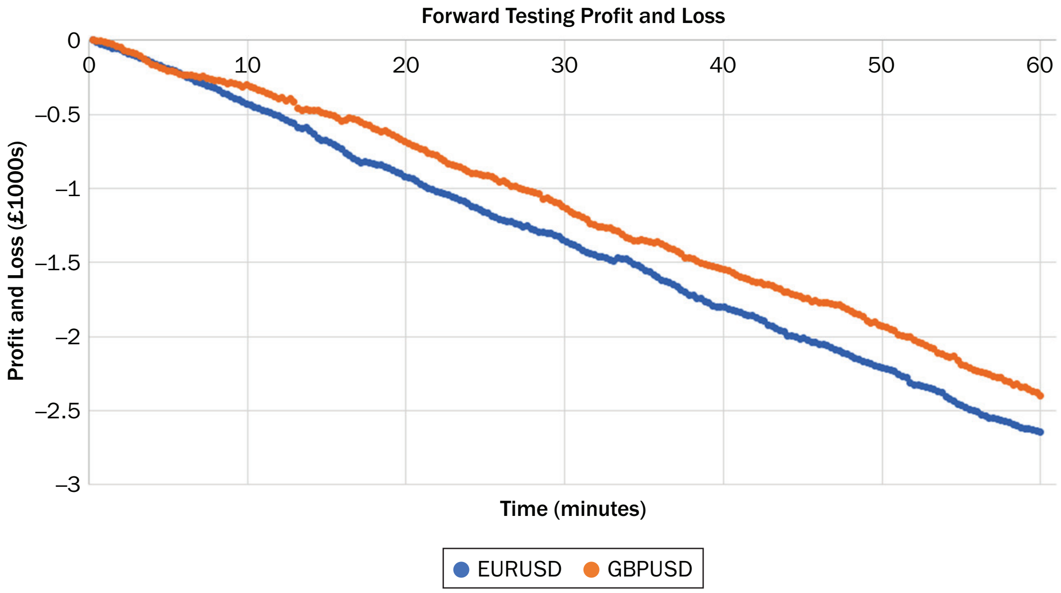
EXHIBIT 23

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

EXHIBIT 24

Forward Testing Results Using Q-Learning on GBPUSD and EURUSD



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 alternative 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.

Explainable AI

This part 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 long to short. Exhibits 25, 26, and 27 show heatmaps representing the average approximated Q-values normalized 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), that is, 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.

EXHIBIT 25

Q-Value Approximates for Q-Learning Table (averaged across all instruments)

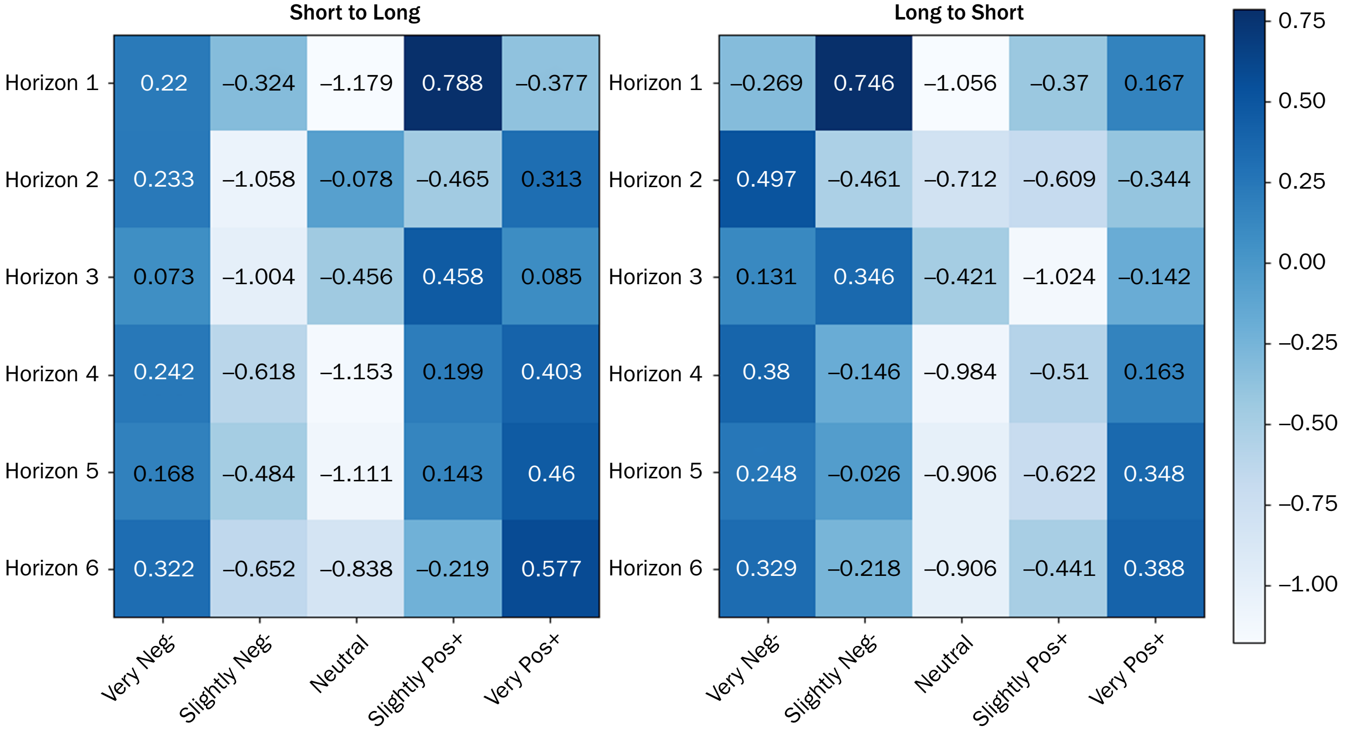


EXHIBIT 26

Q-Value Approximates for DQN (averaged across all instruments)

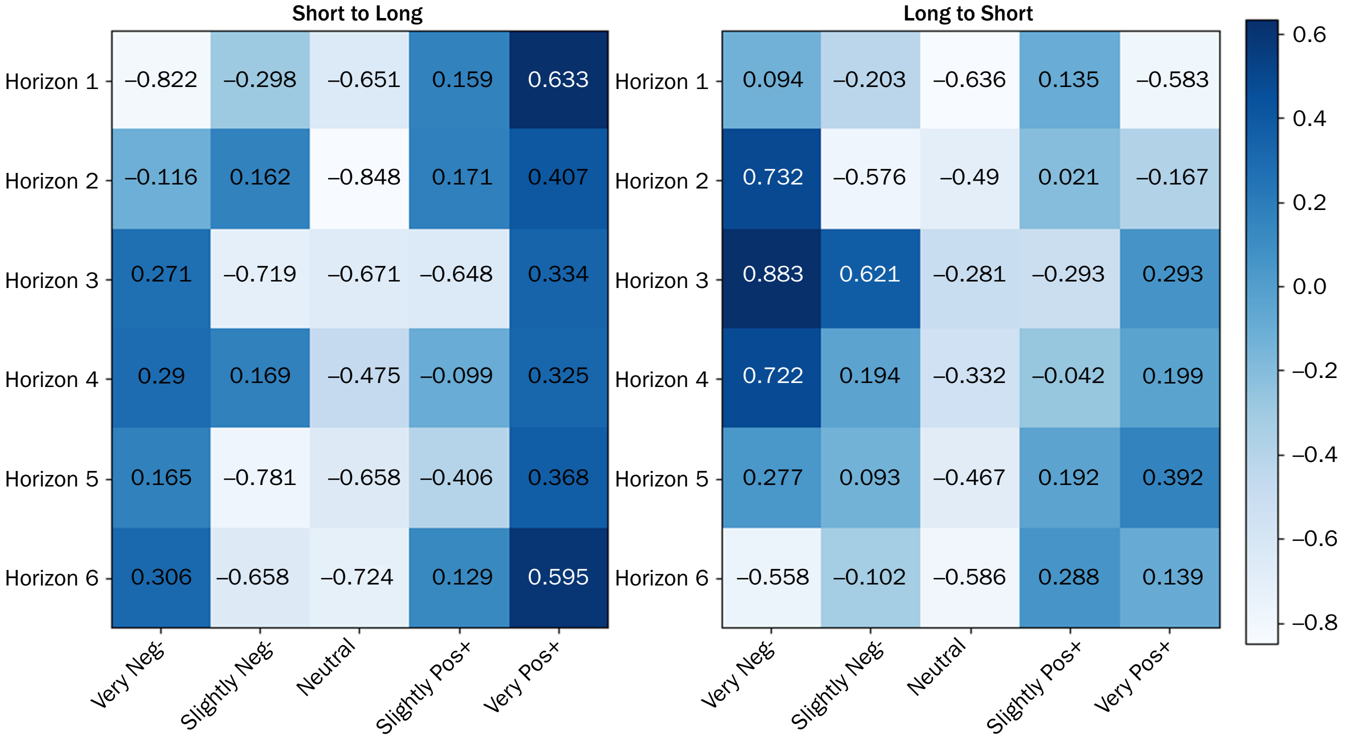
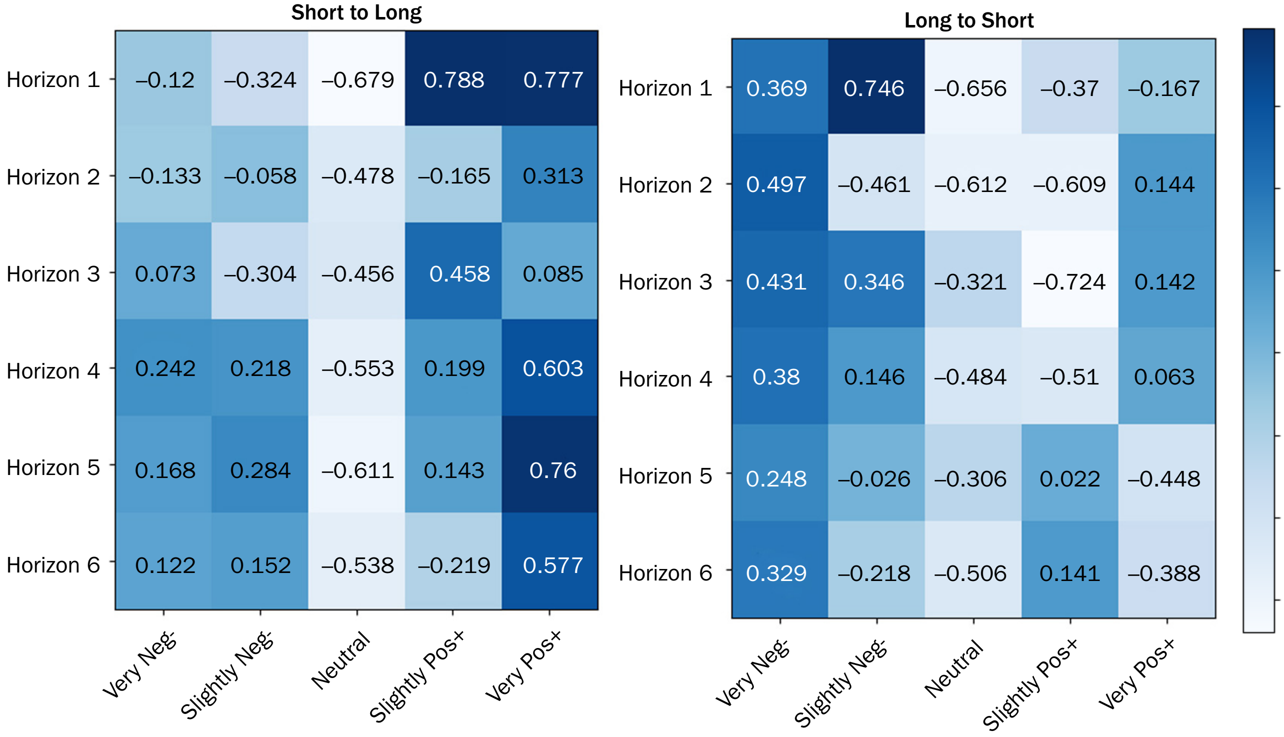


EXHIBIT 27

Q-Value Approximates for DDQN (averaged across all instruments)



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.

CONCLUSIONS AND FURTHER WORK

This research sets out to combine deep learning on the order books with RL 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 time steps, 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, Turiel, and Westray's (2021) results using the MLP architecture, and the rankings among the agents align with Bertermann's (2021) findings. Only 10 weeks of OFI 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 time steps, 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 standardize 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 end-user license agreement because this work is for research and not commercial gain (CTrader 2023). This issue was minimized by only storing wanted OFI features inferred from the raw limit order book states instead of storing actual raw states directly.

APPENDIX

EXHIBIT A1

Tabular Performance Comparison of Q-Learning and DQN Provided in Bertermann (2021)

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 1,400–1,500	2.0687	0.4418	39.61	0.052
Episodes 2,000–2,100	2.1129	0.3542	39.09	0.0541
Episodes 2,400–2,500	2.1381	0.3956	38.64	0.0553
DQN				
Episodes 0–100	–0.0061	0.4916	49.15	–0.0001
Episodes 400–500	0.6417	0.5354	38.74	0.0166
Episodes 1,400–1,500	1.1383	0.7223	21.89	0.0520
Episodes 2,000–2,100	1.4255	0.6826	21.79	0.0654
Episodes 2,400–2,500	1.5758	0.6206	20.98	0.0751

REFERENCES

Auquan. “Evaluating Trading Strategies.” *Medium*, 2017. https://medium.com/auquan/evaluating-trading-strategies-fe986062a96b.

Bertermann, A. 2021. “Reinforcement Learning Trading Strategies with Limit Orders and High Frequency Signals.” Imperial College London.

Billiet, P. 2003. “The Mann-Whitney *U*-test—Analysis of 2-Between-Group Data with a Quantitative Response Variable.” University of Nebraska-Lincoln.

Breckenfelder, J. 2020. “How Does Competition among High-Frequency Traders Affect Market Liquidity?” European Central Bank.

ByBitHelp. “Introduction to TWAP Strategy.” 2023. https://www.bybithelp.com/en-US/s/article/Introduction-to-TWAP-Strategys.

CMC Markets. “Mean Reversion Trading Strategies.” 2023a. https://www.cmcmarkets.com/en-gb/trading-guides/mean-reversion.

——. “Momentum Trading Strategies.” 2023b. https://www.cmcmarkets.com/en-gb/trading-guides/momentum-trading.

Cont, R., A. Kukanov, and S. Stoikov. 2014. “The Price Impact of Order Book Events.” *Journal of Financial Econometrics* 12 (1): 47–88.

CTrader. “End-User License Agreement.” 2023. https://ctrader.com/eula/.

CTrader FXPro. 2023. https://www.fxpro.com/.

Huang, R., and T. Polak. 2011. “LOBSTER: Limit Order Book Reconstruction System.” *SSRN* 1977207.

Karpe, M., J. Fang, Z. Ma, and C. Wang. 2020. “Multi-Agent Reinforcement Learning in a Realistic Limit Order Book Market Simulation.” In *Proceedings of the First ACM International Conference on AI in Finance*, Association for Computing Machinery. New York, NY, pp. 1–7.

Kohout, J. 2022. “Volume Delta Reversal Trade Strategy.” https://axiafutures.com/blog/volume-delta-reversal-trade-strategy.

Kolm, P. N., J. Turiel, and N. Westray. 2021. “Deep Order Flow Imbalance: Extracting Alpha at Multiple Horizons from the Limit Order Book.” *SSRN* 3900141.

Köppen, M. “The Curse of Dimensionality.” In 5th *Online World Conference on Soft Computing in Industrial Applications*, vol. 1, pp. 4–8, 2000.

Lahmiri, S., and S. Bekiros. 2021. “Deep Learning Forecasting in Cryptocurrency High-Frequency Trading.” *Cognitive Computation* 13 (2): 485–487.

Mäkinen, Y., J. Kanniainen, M. Gabbouj, and A. Iosifidis. 2019. “Forecasting Jump Arrivals in Stock Prices: New Attention-Based Network Architecture Using Limit Order Book Data.” *Quantitative Finance* 19 (12): 2033–2050.

Mani, M., S. Phelps, and S. Parsons. 2019. “Applications of Reinforcement Learning in Automated Market-Making.” In *Proceedings of the GAIW: Games, Agents and Incentives Workshops.* Montreal, Canada, 13–14.

——. 2006. “Reinforcement Learning for Optimized Trade Execution.” In *Proceedings of the 23rd international conference on Machine learning*, pp. 673–680.

Nvidia. CUDA. 2023. https://developer.nvidia.com/cuda-zone.

Passalis, N., A. Tefas, J. Kanniainen, M. Gabbouj, and A. Iosifidis. 2020. “Temporal Logistic Neural Bag-of-Features for Financial Time Series Forecasting Leveraging Limit Order Book Data.” *Pattern Recognition Letters* 136: 183–189.

Pytorch. 2023. https://pytorch.org/.

Singh, A., and R. Pachanekar. “Sharpe Ratio: Calculation, Application, Limitations.” QuantInsti, 2019. https://blog.quantinsti.com/sharpe-ratio-applications-algorithmic-trading/#Sortino.

Spooner, T., J. Fearnley, R. Savani, and A. Koukorinis. “Market Making via Reinforcement Learning.” In *Proceedings of the 17*th *International Conference on Autonomous Agents and Multi Agent Systems*, pp. 434–442, 2018.

TradingFX VPS. 2023. https://www.tradingfxvps.com/.

Tran, D. T., A. Iosifidis, J. Kanniainen, and M. Gabbouj. 2019. “Temporal Attention-Augmented Bilinear Network for Financial Time-Series Data Analysis.” In *IEEE Transactions on Neural Networks and Learning Systems* 30 (5): 1407–1418.

Vyetrenko, S., and S. Xu. 2019. “Risk-Sensitive Compact Decision Trees for Autonomous Execution in Presence of Simulated Market Response.” In *Proceedings of the 36*th *International Conference on Machine Learning.*

Wu, J., M. Siegel, and J. Manion. 1999. “Online Trading: An Internet Revolution.” Massachusetts Institute of Technology.