DEPARTMENT OF COMPUTER SCIENCE

Can Intelligence and Speed Make Profit? A Deep Learning Trader Challenging Existing Strategies in a Multi-Threaded Financial Market Simulation

An exploratory proof-of-concept for next-generation automated trading

Armand Cismaru

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Master of Engineering in the Faculty of Engineering.

Wednesday 3rd May, 2023

Abstract

Advancements in computing have revolutionised the field of Deep Learning which in turn has started to revolutionise everything we rely on. Financial markets are no different. Algorithmic traders are responsible for most of the trades executed today. Millions of trades are being performed asynchronously each day at sub-second rates. Existing literature has been focusing on evaluating trading strategies in simple, minimal simulations of financial markets, overlooking the asynchronous nature of real-world markets.

We propose a new iteration of DeepTrader, a previous approach based on a Deep Learning architecture that has been demonstrated to perform well in sequential simulated markets, to trade in the Threaded-BSE, an asynchronous version of the public-domain market simulator, BSE. We are preserving the original name of the trader to contribute to its public domain development. Our research hypothesis is:

> Is a new high-frequency trader based on a "black-box" Deep Learning model, trained solely on observed behaviour from the LOB of a market capable of competing with and outperforming existing trading agents in an asynchronous, multi-threaded market simulation?

This project dives into the research hypothesis with the intention of giving an exploratory answer within the timeframe and with the resources available for a Master's thesis. The main contributions and efforts of this project are as follows:

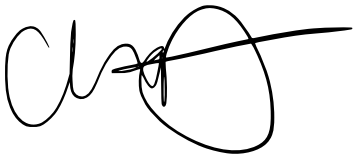
- I spent 100 hours surveying the literature about trading strategies, simulations, and Deep Learning systems applied in finance.
- I developed an intelligent trading strategy capable of training on financial data and producing profit in simulated markets.
- I extended a 3500-line Python system to generate the required data and enable the testing of our new system.
- I developed cloud infrastructure capable of running large-scale simulations, running an estimated 21.000 market sessions with an accumulated time duration of 350 hours.
- I evaluated the new strategy against existing trading algorithms and showed that it outperformed in most of the experiments.

Dedication and Acknowledgements

This project is the "crown jewel" of my four years of study at the University of Bristol. It was made possible by the great advice and guidance of my supervisor, Dr. David Cliff. I want to thank my family for their love and hard work to support me throughout my studies, the friends who encouraged me when most needed and my lovely girlfriend Andreea, my biggest fan, for her love, support and great food. I also want to acknowledge the staff at Pret A Manger on Queens Rd Bristol for their great coffee that fueled this project.

Declaration

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Taught Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, this work is my own work. Work done in collaboration with, or with the assistance of others, is indicated as such. I have identified all material in this dissertation which is not my own work through appropriate referencing and acknowledgement. Where I have quoted or otherwise incorporated material which is the work of others, I have included the source in the references. Any views expressed in the dissertation, other than referenced material, are those of the author.



Armand Cismaru, Wednesday 3rd May, 2023

Contents

List of Figures

List of Tables

Ethics Statement

This project did not require ethical review, as determined by my supervisor, Dr. David Cliff.

Supporting Technologies

This section presents a summary of the technologies and third-party software that this project is using, as follows:

- The codebase of this project is written in Python 3.4 or above.
- Used the EC2, EKS, and S3 cloud services provided by AWS to build the cloud infrastructure required.
- The Python Package Installer PIP (23.1) was used to install any Python dependencies.
- Kubernetes version 1.26.3 and its command line interface, kubectl, were used to manage the cloud compute clusters.
- Docker and the Docker Image Repository were used to turn our system into a portable image that could be run as a container on any Linux machine.
- GitHub was used as the remote repository for this project and Git as the version control software.
- GitHub Actions were used to configure a workflow containing a job running pylint (2.17.2) for static code checks.
- The project uses the following dependencies: Boto3 (1.26.117), Numpy (1.24.2), Matplotlib (3.7.1), Keras (2.12.0), Tensorflow (2.11.0), Pandas (1.5.3) and Seaborn (0.12.2).
- The Threaded Bristol Stock Exchange, developed by Cliff and Rollins, was used as a test bed for generating the data and evaluating our trading strategy for this project.

Notation and Acronyms

Notation and acronyms used across this paper are as follows:

AA	:	Aggressive Adaptive
ABM	:	Agent Based Modelling
AI	:	Artificial Intelligence
ANN	:	Artificial Neural Network
AWS	:	Amazon Web Services
BC4	:	Blue Crystal Phase 4
BPTT	:	Backpropagation Through Time
BSE	:	Bristol Stock Exchange
CDA	:	Continuous Double Auction
CNN	:	Convolutional Neural Network
CSV	:	Comma-Separated Values File Format
DL	:	Deep Learning
DLNN	:	Deep Learning Neural Network
EC2	:	Elastic Compute Cloud
EKS	:	Elastic Kubernetes Service
FIFO	:	First In First Out
GDX	:	Gjerstad & DickhauT eXtended
GPT-3	:	Generative Pretrained Transformer 3
GPU	:	Graphical Processing Unit
GUI	:	Graphical User Interface
LSTM	:	Long Short-Term Memory
LOB	:	Limit Order Book
ML	:	Machine Learning
MSE	:	Mean Square Error
NYSE	:	New York Stock Exchange
ReLU	:	Rectified Linear Unit
RNN	:	Recurrent Neural Network
SDK	:	Software Development Kit
S3	:	Simple Storage Service
TBSE	:	Threaded Bristol Stock Exchange
YAML	:	YAML Ain't Markup Language
ZIC	:	Zero Intelligence Constrained
ZIP	:	Zero Intelligence Plus
ZIU	:	Zero Intelligence Unconstrained
	:	
Smith's α	:	Metric used for price volatility estimates, see Section 2.4
α	:	Significance level value for statistical testing
μ	:	Learning rate used to train the model
$p - value$	:	A statistical measurement used to validate a hypothesis against observed data
σ	:	Standard deviation
$\bar{x}$	:	Mean of x individual profit recordings

Chapter 1

Introduction & Contextual Background

1.1 Introductory notes

Financial markets sit at the core of modern economics, enabling individuals, companies, or countries to exchange commodities such as gold and oil, derivatives such as futures and options, or securities such as stocks and bonds, among others. The way most of them operate is based on a type of auction. Most of us imagine the idea of an auction being similar to what is happening at an art sale: people bid a price they are willing to pay, and the highest bid wins. That system is known as the English auction. Financial markets operate on what is called a Continuous Double Auction (CDA) [16], a type of auction where the buyers and the sellers continuously adjust their prices in order to find a common ground for trades. CDAs make use of a Limit Order Book (LOB), a shared ledger of bids and offers placed by traders. LOBs are of central importance to a lot of global exchanges and, subsequently, to our project.

The participants in a financial exchange are called traders—agents that engage in short-term buying or selling of financial assets for themselves or on behalf of a client [22]. Their sole purpose is to maximise profit: the difference between the limit price that a seller or buyer might ask or bid for and the price at which they trade. A market is made up of tens of thousands of traders that operate synchronously under continuously changing market conditions. Usually traders are issued orders from their clients, and they charge a commission for the profit they make. How well a trader performs in a market relies in its ability to observe market changes and adapt its strategy accordingly. Algorithmic traders represent pieces of software that are enabled to trade on their own using a pre-defined algorithm, having now (almost) replaced human traders. They have the ability to analyse more data and take faster decisions than any human would ever be able to.

Lately, the rise of Artificial Intelligence has opened up a new world of possible applications. From chatbots to cancer research, AI is used to fill the gaps that humans cannot. Deep Learning Neural Networks (DLNNs) represent a form of AI designed after the human brain [15], and have been applied in various fields, including speech recognition, computer vision, natural language processing, and healthcare [2][36], with prior literature demonstrating that DLNN-based traders can match or outperform existing algorithmic traders [9]. More importantly, the same democratisation of computing power that led to the explosion of DLNNs has enabled scholars to develop more realistic market simulations, opening up lots of research opportunities in this field.

The purpose of this project is to bring together two topics of research in this area by extending and enhancing a proven DLNN algorithmic trader to trade in a multi-threaded simulation. The agent would be trained on historical LOB data generated by the same simulation that it would later trade in. The aim is to build and tune a system that could outperform existing strategies, a result which would raise questions such as: What is the actual purpose of not investing all energy into superior universal AI

traders?

A positive result could have value in the real world, with the only barrier being the ease of access to some of the LOB data that it requires. Customer limit prices are not public when trading, so having access to that kind of historical data would prove to be great leverage. In the case of a negative result, this research would prove useful to investigate its underlying causes and ascertain the vulnerabilities that a DLNN trader has when deployed in a realistic simulation. This project is inspired by and extends the efforts of two pieces of research: the work on the first version of DeepTrader by Cliff, Meads, and Wray [42] and the development of the Threaded Bristol Stock Exchange by Cliff and Rollins [28].

1.2 Motivation

1.2.1 Modern economics

Economics affects every aspect of our lives, visible or not. Economics dictate how and where resources are allocated and how accessible they are to us, the ultimate users. Trade is the steam that makes the engine of the economy move. Without it, resources wouldn't be able to make it from their production site to the end user, preventing the existence of national and trans-national supply chains, increasing the chance of resource waste, and decreasing competitiveness, hence higher prices.

Financial exchanges enable global trade, with millions of interactions happening at once each day of the year. They are, essentially, the engine of the global economy. But they are far from perfect and have flaws. Markets are very sophisticated and complicated, so these systems require a large number of specialists to manage and regulate them.

As AI technology continues to advance, it is increasingly being adopted in financial exchanges. Technology has empowered the creation of modern markets, and technology will help ensure fairness and efficiency in the future. AI-algorithmic traders have the potential to increase the efficiency and accuracy of financial exchanges by processing more data and making faster and better-informed trading decisions. Risk management can benefit from the AI traders' ability to detect anomalies and predict market trends. Extensive research can help regulators understand their behaviour and prepare the legal framework in order to ensure compliance and fairness.

1.2.2 Computer Science

Computers are quite a recent invention, created with the purpose of externalising tasks that we humans don't want or are not able to do. Step by step, we have become dependent on their abilities. Computing power, until very recently, was expensive and inaccessible. In time, as computers became more and more accessible, the number of applications built on top of them grew exponentially. At the same time, the energy required to build and optimise software has grown at the same rate.

In Computer Science, a problem might never have a final solution, as there will always be new ways to optimise it. Deep Learning, a subset of Artificial Intelligence, is a type of learning modelled after the neural connections in the human brain. It is able to ingest vast amounts of data with the purpose of optimising its objective function. In essence, it is a game of balancing resource allocation (computing, time) and performance. Deep Learning can find better solutions to a resource-consuming problem by consuming resources itself.

The efficiency and accessibility of computing power mean that, for the problem of building a Deep Learning trader agent trained in realistic simulations, we are able to perform that with low cost and in reasonable timeframes. On-demand cloud computing can be used to generate our data, and highly optimised GPUs such as those in the Blue Crystal 4 can train the network in under 24 hours. Creating such a trading agent on real-life financial exchanges can reduce the number of failed trades, optimise

the trading process, and help mitigate risk. All these factors contribute to a more sustainable financial industry by reducing resources and costs.

1.2.3 Aiming for realism

Trying to model a real financial exchange can be incredibly difficult given how complex it is. This is the reason why market simulations used for research are simplified versions of real financial exchanges, designed to include a number of customizable conditions but simple enough to be manageable and efficient. One such simulation is the Bristol Stock Exchange (BSE) [12], which TBSE is based on. Created by Cliff, it has been continuously evolving over the past decade, becoming a well-respected financial market simulation enabling the trading of one type of fictional commodity. BSE does not include the issue of latency, missing out on the concurrent nature of real markets. In order to be useful for real-life applications, research has to come as close to reality as possible. As we will discuss in the next section, there is a gap in the literature on studying AI traders in parallel simulations that this project intends to address.

1.3 Previous work

1.3.1 Beginnings of experimental economics and agent based modelling

The groundwork for experimental economics was laid by Vernon Smith in 1962 by publishing "An Experimental Study of Competitive Market Behaviour" in The Journal of Political Economy (JPE)[33]. Smith has implemented a series of experiments based on the Continuous Double Auction (CDA) system, where buyers and sellers are announcing bids and offers in real-time, with the possibility of a trade being executed any time the prices match.

The experiments were performed with small groups of human traders. They were instructed to trade an arbitrary commodity on an open-pit trading floor with the intention of maximising profitability, namely the difference between the limit price and the trade price. Each trader was given a pre-defined limit price: for sellers, the minimum they are allowed to sell their units at, and for buyers, the maximum price they can pay for a unit of the traded asset, thus preventing loss-making trades. The simulations were carried out as "trading days", namely time intervals of 5 to 10 minutes. The quotes that were shouted by the traders resembled the Limit Order Books of modern markets. Once a trader agreed on a trade with its counterparty, both would leave the market as they only had a single unit to trade.

The results showed rapid convergence to the theoretical equilibrium price, measured by Smith's "alpha" (α) metric. It measures how well and efficiently the market is converging to the equilibrium price. The experiments capture the asynchronous nature of financial markets, one of the issues that this work is aiming to explore. Vernon Smith received the Nobel Prize in 2002 for his pioneering work in experimental economics, with his experiment styles being the basis of most research carried out in this field and the methodology used in this paper.

Three decades later, in 1993, Gode and Sunder introduced the Zero Intelligence traders [19]. Their focus is studying how automated traders perform in markets dominated by human traders. They introduced two trading strategies: Zero Intelligence Unconstrained (ZIU) and Zero Intelligence Constrained (ZIC). ZIU is generating purely random quotes while ZIC is limited, constrained to a price interval. Their experiments carried out in the style of Vernon Smith showed ZIC to outperform human traders. A few years later in 1997, Cliff has published a paper proposing Zero Intelligence Plus (ZIP) traders, which, by using a simple form of Machine Learning, can be adaptive and converge in any market condition [11]. ZIP is based on a limit price and an adaptive profit margin. The margin is influenced by a learning rule and the conditions of the market.

In 1998, Gjerstad & Dickhaut described an adaptive agent, GD [18], with Tesauro & Bredin publishing

a paper in 2002 describing the GD eXtended (GDX) trading algorithm [38]. In 2006, Vytelingum's thesis introduced what is called the Aggressive-Adaptive (AA) strategy [40], which was thought to be the best-performing agent until recently. In 2019, Cliff and Snashall performed comprehensive experiments comparing AA and GDX, simulating over a million markets. The results show that AA is routinely outperformed by GDX, arguing that advancements in cloud computing and compute power open new possibilities for strategy evaluation that were not possible before [34]. Rollins and Cliff's (2020) work on the Threaded BSE, which we are going to use, also demonstrates a new dominance hierarchy by better simulating the asynchronous and time-dependent nature of real-life markets [28].

1.3.2 Rise of intelligence in market modelling and price prediction

The advent of AI has attracted increased attention in the fields of finance and trading. More and more papers detail how advanced Deep Learning methods became very powerful tools in the world of agent-based trading, market making, and price forecasting. Axtell and Farmer present, in their 2018 report, how advances in computing agent-based trading (ABM) have impacted how trading is performed today [6]. In finance, ABM helped us understand markets, volatility, and risk better. Their report is comprehensive and can be considered a higher-level point of reference on how agents are being applied in different branches of finance and economics. Njegovanović published a paper in 2018 that discusses the implications of AI and ML in finance, with a focus on how the human brain and its behaviour have inspired the architecture of automatic decision models [26].

In the past decade, a number of studies have explored the potential of Machine Learning and Deep Learning in finance. In 2013, Stotter, Cartlidge, and Cliff introduced a new method for assignment adaptation in ZIP, performing balanced group tests against the well-known ZIP and AA strategies [35]. Their results show that assignment-adaptive (ASAD) traders equilibrate more quickly after market shocks than base strategies. In 2019, Ji, Kim, and Im performed a comparative study of DNN vs. LSTM for Bitcoin price prediction, trained on historical data from public ledger records. They conclude that classification models (DNN) perform better than regression models (LSTM) for price prediction [23].

Another paper from 2020 by Silva, Li, and Pamplona uses LSTM-based trading agents to predict future trends in stock index prices. Their proposed method, named LSTM-RMODV, demonstrates the best performance out of all studied methods, and it is shown to work in both bear and bull markets [31]. Deep Reinforcement Learning was found to be performant in market-making applications by Sun, Huang, and Yu in 2022 [37], with a similar piece of work being published in 2019 by Sirignano and Cont [32]. They propose a Deep Learning model applied to historic US equities markets. The information extracted from the Limit Order Books uncovers a relationship between past orders and the direction of future prices. They conclude that this is better than specialised predictions for specific assets. Their results illustrate the applicability and power of Deep Learning methods in modelling market behaviour and generalisation.

1.3.3 Need for intelligence and realistic modelling

The work in this paper continues what Calvez and Cliff started in 2018 [9], when they introduced a Deep Learning Neural Network (DLNN) system trained to replicate adaptive traders in a simulated market. Purely based on the observation of the best bid and ask prices, the DLNN has managed to perform better than the trader observed. In 2020, Wray, Meades, and Cliff will take this further by introducing the first version of DeepTrader, a high-performing algorithmic trader [42] trained to perform in a sequential market. Based on a LSTM, it automatically replicates a successful trader by training on 14 features derived from Level-2 market data. The first version of DeepTrader matches or outperforms existing trading algorithms in the public-domain literature.

Most studies are performed on sequential simulations, in which the speed at which the traders react to changes in the market does not matter. Axtell and Farmer argue, in their report "Agent-based modelling in economics and finance: Past, present, and future", mentioned above [6], that the real social and economic worlds are parallel and asynchronous, but we try to replicate it with single-threaded code. Rollins and Cliff try to mitigate this in a paper they published in 2019 [28]. They propose a threaded

market simulation based on BSE, on which they perform pair-wise experiments between well-known trading strategies. The results reported very interesting insights, with a new dominance hierarchy of trading algorithms emerging, opening a new area of research. Our work aims to build on the Threaded-BSE and the architecture of DeepTrader, questioning how this new system behaves in parallel simulation, thus bringing more insight into how it would perform in the real world.

1.4 Evaluation of Current State of Financial Markets and AI

Humans have been trying to explain, replicate, and quantify what it means to be intelligent for hundreds of years. The theoretical foundations of Machine Learning and Deep Learning were laid in the 1960s [30] but have taken off in the past decade as a result of advancements in computing power and reduced costs [3]. Finance has been at the core of human activity since its beginnings. Agent-based trading systems are a modern invention, with studies on experimental economics and auction dynamics having started just over 50 years ago, pioneered by Vernon Smith in 1962, who published "An experimental study of competitive market behaviour" in The Journal of Political Economy (JPE) [33]. The project proposed here continues the natural flow of evolution by using AI in advanced simulators that better reflect the realities of contemporary financial markets.

Advancements in computing have revolutionised the field of DL, which in turn has started to revolutionise everything we rely on. Financial markets are no different. Algorithmic traders are responsible for most of the trades executed today. A report by Acumen Research and Consulting estimates that the algorithmic trading market size will grow from $14.1 billion in 2022 to $41.9 billion by 2030 [4]. Millions of trades are being performed asynchronously each day at sub-second rates. Market orders are being executed at the same time by competing traders looking to maximise profit. They compete against each other, with each nanosecond being pivotal in getting the best deal in markets with constantly changing supply and demand based on external events. A key issue is that existing literature has focused on evaluating trading strategies in simple, minimal simulations of financial markets, overlooking the asynchronous and volatile nature of real-world markets. This was the case with previous work involving DeepTrader.

From medicine discovery to generative models like GPT-3 [17][8], DL systems have demonstrated the power they can leverage when trained and used in appropriate environments. The extensive literature review performed prior to this project found that DL algorithmic traders have been successfully used in financial market simulations in this research field. They have the capacity to replicate successful traders and quickly react to market changes better than humans and other well-established public-domain trading strategies, but there has not been enough emphasis on trying to capture the dynamics of real markets. This project intends to bridge this gap by proposing a powerful DL system that trains in a realistic setting, creating a reliable platform for testing AI-based trading agents with big potential benefits for real financial markets.

1.5 Project approach

One of the central challenges of this project is building on and integrating existing work to create a novel, high-performance system that enables both training and testing of a Deep Learning trader. Past approaches have demonstrated that a simple architecture can match or even outclass existing strategies, but only when trained on large amounts of data [9][42][44]. This is an issue when trying to acquire data that reflects real-life markets, as this might be hard or expensive to find.

The issue of large amounts of realistic data required for a Deep Learning model is addressed by using the Threaded-BSE as the test bed of this research project. Threaded-BSE is designed to be a versatile and easy-to-deploy market simulation that uses historical real-world data to generate supply and demand schedules. TBSE is based on a LOB, making it easy to extract the feature information needed for our model. The market simulation implements a number of known trading strategies, which will be used in different settings and conditions to generate useful training data.

The concurrent design of TBSE has already produced results that question the hierarchy of previously known strategies, so another challenge that we will be facing is the need for rigorous evaluation of results that are novel without a solid reference for ground truth. This will be done by performing statistical tests on hundreds of simulations for each pair of strategies, determining if the AI agent is dominating or being dominated, and the conditions and causes for such results.

The high-level objective is to create an intelligent trading agent that is able to challenge existing public-domain traders in a realistic simulation, addressing a gap in the relevant architecture. Whether the Deep Learning model will be more profitable or not depends on a variety of factors, but regardless of the result, this project will pave the way for further research and development in this area.

The roadmap for this project is as follows:

1. Perform extensive research on existing literature on Artificial Intelligence in finance, with a focus on automated trading agents.
2. Build on the knowledge accumulated in the past 4 years of the degree to understand, plan, and design every bit needed to assemble this project.
3. Integrate and extend existing code to create a working platform for generating training data and running it in the cloud.
4. Create an automated pipeline for querying, cleaning, and compressing data stored in the cloud in a format ready for training.
5. Train, tune, and improve a profitable new iteration of DeepTrader, optimised for latency-sensitive markets, and test it against existing strategies.
6. Perform a critical evaluation of the performance of the model against existing agents using different methodologies and test for normality and statistical significance.

Chapter 2

Background

2.1 Market Simulations - The field of study vs reality

The research field of finance and modelling relies on huge swaths of data, so much of the focus is centred around acquiring quality data. There are mainly two options: study historical real-world data or use market simulations. The former can be very expensive and restrictive, while the latter requires a lot of computing power, time, and quality software.

As described in Section 1.2.3, TBSE was chosen for its improved realism. But we are far from emulating a real stock market, such as the London Stock Exchange. Building such a complex simulation requires years of work and vast resources. For example, having the ability to trade more than one commodity would be an immense asset and would open up a lot of research hypotheses, but it would require serious refactoring and extension of the existing software, work beyond the scope of this project. In order for research to be useful, you need to be able to isolate and reproduce certain conditions, so a more minimalistic approach is better in this regard.

2.2 Continuous Double Auctions & The Limit Order Book

As first described in Section 1.1, most financial exchanges rely on what is called the Continuous Double Auction (CDA).

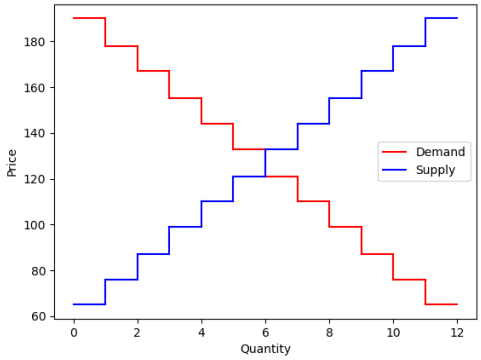
The CDA is a type of auction that merges the English and Dutch auction styles to match buyers and sellers efficiently. In an English auction, the auctioneer sets an initial price, and buyers compete against each other by offering higher bids until the highest one wins. Conversely, in a Dutch auction, the auctioneer sets a high asking price and gradually lowers it until a bidder accepts it, concluding the auction [25]. By combining these two auction styles, the CDA matches buyers with sellers and ensures that supply and demand curves cross at an equilibrium price, which is the price that the financial asset is being traded at.

In many CDAs, the Limit Order Book (LOB) is the central component. This is a shared data structure that contains bids and asks posted by traders while maintaining anonymity. The LOB is divided into two sections—one for buyers and one for sellers—with each side indicating the price and quantity that the buyer or seller is willing to trade. The highest bid price is listed at the top of the buyers' side, while the smallest ask price is listed at the top of the sellers' side, with prices being sorted in ascending order.

In financial market terminology, when a trader wants to sell at the current best bidding price, that is called "hitting the bid". Conversely, when a trader wants to buy at the best asking price, we refer to it as "lifting the ask". The difference between the best bid and the best ask is known as the "spread", so

the aforementioned situations being mentioned as "crossing the spread".

Additionally, financial exchanges maintain a "tape", a time-series record of orders on the exchange. This log includes the timestamp, price, and identification of the counterparties involved in the trade. Each trader also updates their own "blotter", a local record of the trades the agent is involved in.



(a) Symmetric supply and demand curves for an asset.

XYZ			
Bid		Ask	
10	152	155	20
60	150	162	50

(b) Graphical visualisation of a simple LOB.

Figure 2.1: Market curves and LOB view. [1]

Figure 2.1 contains, on the right-hand side, what a LOB would look like in the GUI of a trading system. It shows the prices and quantities of any outstanding orders that the originating traders have not cancelled. In this case, we can observe the XYZ asset being priced, as professional traders might refer to it, at "$152 - 155$". The data in a LOB is referred to as "Level-2" data, the kind of data that TBSE produces and will be required for our model. For context, "Level-1" market data contains only the prices and quantities of the best bid and ask in the market.

The left-hand side of Figure 2.1 represents a supply and demand schedule, in this case a symmetric one in which all traders get their customer orders at the same time. The quantity axis means that, for a market with 12 total traders, an N number of traders are able to trade at a fixed price. For example, at quantity 2, only two traders would be able to buy at a price over 180, while only two would be able to sell at a price less than 80. The curves meet at the theoretical equilibrium price, where supply and demand are balanced by market competition.

2.3 The Threaded Bristol Stock Exchange

The Threaded Bristol Stock Exchange (TBSE) is an asynchronous market simulation based on the Bristol Stock Exchange. It was created by Rollins and Cliff and provides a more realistic environment in which to study trader behaviour and market dynamics [28]. It is based on a CDA, so following the rules laid out by Smith in his Nobel-winning paper, the traders cannot buy or sell at a loss, and their sole purpose is to make profit, namely the margin between their limit price and the price of an executed trade. The main difference between TBSE and the original simulation is the way the market session operates.

In BSE, you would have the session operate in a timed loop, randomly picking a trader at once and asking it if it wanted to place an order. If an order is placed, the change is broadcasted to the other traders, who update their internal records and their strategies subsequently. The simulation then proceeds to randomly pool another trader's order. This implementation does not include the issue of latency, as each trader can take as much real-time as it needs to generate an order or process market changes.

In TBSE, each trader operates its own thread along with the exchange itself. In this loop, lasting for the duration of the market session, each trader receives updates about the trades that have been executed in the market and decides whether to generate an order or not. This way, an algorithm that generates faster quotes has the advantage over one that is stuck updating its internal records. The orders are placed in a

first in, first out" (FIFO) queue, so that the first traders to place their orders will have them processed first, giving advantage to faster strategies and simulating real-life markets.

The orders placed by traders are called limit orders. This means that they agree to buy or sell a specific commodity at a fixed "limit" price or better. TBSE allows transactions at a price of at least 1 and no more than 500. And just like BSE, it is designed to enable the trading of one unit of a single type of financial instrument.

Taking everything into consideration, TBSE is an enhanced simulation that captures the asynchronous nature of real markets and allows the creation of Level-2 LOB data that will be needed for this project.

2.4 Terminology

Over the course of this project, there are a number of specialised terms relevant to our work, especially regarding the LOB. This data structure is the focus of extensive research [40][44][5][10] and provides the information that professional traders use in real life. In order to provide clarity and a point of reference, this section will detail and define (where applicable) a number of financial market terms, as follows:

- The spread - also known as the bid-offer spread and referred to in Section 2.2, is the difference between the highest price p_B a buyer is willing to pay for an asset (the bid) and the lowest price p_A a seller is willing to accept (the ask). The formula for bid-ask spread is defined in Equation (2.1).

$$spread = p_A - p_B \tag{2.1}$$

- The equilibrium price - denoted as p^* and first introduced in Section 2.2, is the price at which the quantity of a good or service demanded by buyers in a market is equal to the quantity supplied by sellers in that same market.

- Profit, sometimes referred to as utility u - is the financial gain that a trader earns by selling or buying an asset at a market price p with respect to its limit price l, defined in Equation (2.2).

$$u = |p - l| \tag{2.2}$$

- Midprice - is the average of the highest bid p_B and the lowest ask price p_A quoted for a financial asset, as defined in Equation (2.3). It is used as the reference price for evaluating the current market value of an asset and can be used by traders who want to buy or sell the asset at a fair price.

$$midprice = \frac{p_B + p_A}{2} \tag{2.3}$$

- Microprice - is a measure of the instantaneous price of a financial asset that takes into account both the bid and ask prices and the quantities available at those prices, with its formula given in Equation (2.4). It is calculated as the weighted average of the bid and ask prices (p_B and p_A), where the weights are proportional to the available quantities q_A and q_B at those prices.

$$microprice = \frac{p_A \times q_B + p_B \times q_A}{q_B + q_A} \tag{2.4}$$

- Limit Order Book Imbalance I - is a measure of the supply and demand imbalance for a financial asset in a LOB. It represents the difference between the total quantity q_B of buy limit orders and the total quantity of sell limit orders q_A at a particular price level in the LOB. The formula for the LOB imbalance is Equation (2.5).

$$I = \frac{q_B - q_A}{q_B + q_A} \tag{2.5}$$

- Smith's α - first referred to in Section 1.3.1, is a metric used to determine the price volatility in a market. It is determined as the root mean square deviation of the prices p_t of the set of trades T around the market equilibrium price p^*. The formula is given in Equation (2.6).

$$\alpha = \sqrt{\sum_{t \in T} \frac{p_t - p^*}{|T|}} \tag{2.6}$$

2.5 Comparing trading strategies

Finding the right methodology for comparing trading strategies is as important as the strategies themselves. The design of the experiments has to be in line with the method of comparing market performance. Traders are dependent on the behaviour of other strategies, so it's essential to study them in a controlled environment where they can compete head-to-head. Drawing inspiration from the work of Tesauro and Das [39], the following options have been considered:

- Homogenous population tests: in this kind of experiment, a market is entirely populated by one type of algorithmic trader. It offers a useful way to perform a holistic evaluation of the dynamic of a market by looking at the allocative efficiency, price variance, or mean profits. This methodology is not so well suited for comparing two strategies in independent trials because of the stochastic nature of TBSE, as each market session involves a level of supply and demand volatility. For this reason, the choice was made not to use it for our project.

- One-to-many tests: in this kind of experiment, a trader is the "defector" out of a homogenous population, using another strategy. This is useful for testing how an algorithm behaves when faced with defection and invasion. It is one of the two test types chosen for our project.

- Balanced group tests: in this experiment design, the buyer and seller populations are evenly split between two types of strategies. The choice of balanced-group tests provides benefits to our research, namely: It is a stochastic-controlled trial method that helps reduce bias sources and improve the internal validity of the study. We want to make sure that the differences observed come from differences between trading strategies, not noise. The tests allow full control of the experiment's conditions. The time frame of the simulation, the supply and demand schedules, and the order interval can all be controlled to isolate the differences between the chosen strategies.

2.6 Algorithmic Trader Agents

The literature review in Section 1.3.1 provided some historical background on automated traders, situating this project in the relevant literature. But in order to provide a comprehensive technical background and provide insight into the strategies our trader is going to compete with, this section will offer detailed information on the strategies implemented within TBSE.

2.6.1 Zero-Intelligence Constrained

In 1993, Gode and Sunder proposed the Zero-Intelligence traders: Zero-Intelligence Unconstrained (ZIU) and Zero-Intelligence Constrained (ZIC) [19]. ZIC was one of the first strategies that claimed to match human traders, as it relies on a simple concept to generate quote prices: it generates a random price within the interval bounded by the worst bid on the LOB and its own limit.

2.6.2 Giveaway

Giveaway is a trader agent adapted from Gode and Sunder's ZIU trader. Unlike ZIU, which would generate completely random quotes, even trading at a loss, Giveaway just trades at its own limit price.

2.6.3 Zero Intelligence Plus

Motivated by the fact that the Zero Intelligence agents trade at significantly different prices than the theoretical equilibrium level, failing to equilibrate the market due to their stochastic nature, Cliff proposed an adaptive approach, the Zero Intelligence Plus (ZIP) [11]. ZIP is based on an elementary form of machine learning, adapting its own strategy over time. It responds to changes in the market by altering its profit margin M by using only the information available on the LOB. The adaptation of ZIP is based on the formula in Equation (2.7), generating a price P_i for a unit i based on a limit price L_i and a profit margin M_i.

$$P_i = L_i \times (1 + M_i) \tag{2.7}$$

The formula implies that the margin $M \geq 0$ for buyers is raised by increasing M and lowered by decreasing M. For the seller, the margin $M \leq 0$ is raised by lowering M and lowered by increasing M. In order to remain competitive in the market, a ZIP trader must adapt its margin M based on the actions of other traders, following a learning rule. This is implemented by using one of the most simple machine learning rules, the Widrow-Hoff "Delta Rule". By working in this way, ZIP has achieved performance significantly closer to humans than Gode and Sunder's ZIC trader.

2.6.4 Gerstadt Dickhaut eXtended

In a 2002 paper titled "Strategic Sequential Bidding in Auctions using Dynamic Programming", Tesauro and Bredin propose a new trading strategy, the Gerstadt Dickhaut eXtended (GDX) [38]. GDX is based on a Dynamic Programming framework in which states are represented by the agent's holdings. A "belief function" f, along with a time-series forecast on how it changes over time, is computed as a state-transition model, which solves the DP each time a trade is requested from the GDX agent. The function is used to estimate the probability that a bid at price p for a single unit of a commodity will be accepted.

The "belief function" is based on the history H of market activity and calculated using evidence of prior trade likelihood. For example, a seller would use historical events providing positive evidence to calculate its belief function for an ask by including: $AAG(p)$—the number of accepted asks with a price $\geq p$; and $BG(p)$—any bids with a price $\geq p$. On the other hand, it will use negative evidence of trade likelihood by including $UAL(p)$—unaccepted asks with price $\leq p$. The formula for calculating the belief function $f_s(p)$ for a seller s is given in Equation (2.8).

$$f_s(p) = \frac{AAG(p) + BG(p)}{AAG(p) + BG(p) + UAL(p)} \tag{2.8}$$

For prices not yet in H, the belief function $f(p)$ returns a cubic-spline interpolation between the knot points immediately greater and less than p. Having $f(p)$, the agent generates a quote price designed to maximise surplus μ, the product between the value of $f(p)$ and the utility u, as defined in Equation (2.9).

$$\mu = f(p) \times u \tag{2.9}$$

2.6.5 Adaptive Aggressive

Adaptive Aggressive, or AA, is a trader agent designed by Vytelingum for his PhD thesis in 2006 [40]. For years, it was considered the best trading strategy in the literature [14], until recently. In 2019, Snashall and Cliff question the status quo, with new results resulting from over a million market simulations proving that AA is outperformed by GDX [34]. Additionally, the results obtained by Rollins and Cliff in the paper that introduces TBSE suggest that ZIP might be the dominant strategy in this new environment [28].

AA is constantly adapting to dynamic market conditions and changing competitive market equilibrium. The agent uses an estimate p^* of the competitive equilibrium to calculate its strategy. Equation 2.10 describes how p^* is calculated using a weighted average of the N most recent transaction prices.

$$\hat{p}^* = \frac{\sum_{i=T-N+1}^{T} w_i p_i}{\sum_{i=T-N+1}^{T} w_i}, where\ w_T = 1\ and\ w_{i-1} = \lambda w_i \tag{2.10}$$

AA uses an aggressiveness model; thus, a trader is called "aggressive" when it trades profit for a higher chance of transacting by placing better offers than what it believes the competitive equilibrium price to be. A "passive" trader does the opposite, submitting worse offers than its estimate $\hat{p}^*$, going for higher profits rather than more transactions. Lastly, a trader is "neutral" when it places orders at the competitive equilibrium price. The level of aggressiveness is given by $r \in [-1, 1]$, with $r < 0$ being aggressive, $r = 0$ neutral, and $r > 0$ passive.

In order to generate its target price, AA considers traders to be intra-marginal if they are buyers or sellers and have a limit price higher or lower than the competitive equilibrium price. Conversely, a buyer (seller) is extra-marginal if it has a limit price lower (higher) than $\hat{p}^*$. In normal market conditions, an intra-marginal trader is expected to trade more than its counterpart. This system, combined with the aggressiveness model, defines how a specific trader will behave in a market.

AA uses both long-term and short-term learning rules to adjust its strategy over time and, like ZIP and GDX, is adaptive. How aggressive the agent is gets updated after every new trade on the LOB. Even if recent research indicated that AA might not be the best trading agent anymore, its creation has brought great value to the field and opened the door for further explorations and achievements.

2.7 Deep Learning

First mentioned in the introduction of this project (Section 1.1), Deep Learning (DL) is both a form of Artificial Intelligence and a subset of Machine Learning that uses artificial neural networks to solve complex tasks. It is called "deep" because it involves training neural networks composed of multiple layers, which helps capture patterns in data in a more efficient way. Its main purpose is to train its "objective function" which is used for performing inference on input data.

There are three main types of learning: unsupervised, supervised, and reinforcement learning. For the purpose of our project, we are going to use supervised learning [13]. This involves training a network in the presence of a "teaching signal", labelled data that is used by the model to minimise its training error on the input dataset and create a "mapping" between the input and the output values. In our case, the input features generated from the LOB data are used to predict the mapped value, which is the target price of a trade.

2.7.1 The basics

The first forms of Deep Learning were developed in the 1950s by researchers such as Frank Rosenblatt, who created the perceptron, one of the simplest forms of a neural network [29]. The perceptron is a single-layer neural network that takes in inputs and produces binary outputs based on a set of weights and biases. The perceptron was originally designed for binary classification tasks, such as recognising handwritten digits.

Artificial neural networks (ANNs) are comprised of node layers, each containing an input layer, one or more hidden layers, and an output layer. A node, or artificial neuron, is the smallest computational unit of an ANN.

A node is an activation function that takes in one or more input signals and their corresponding weights, along with a bias term that controls the neuron's overall activation, performs mathematical computation on those inputs, and produces an output signal. A neuron "fires" (or is activated) if the output exceeds a given threshold, then passes the signal to the next layers in the network. Equation (2.11) gives the general formula for a perceptron that takes in a set of inputs x, their corresponding weights w, and the bias term b and produces an output y. Figure 2.2 illustrates a perceptron network with a single layer.

$$y = f(\sum_{i=1}^{n} w_i x_i + b) \tag{2.11}$$

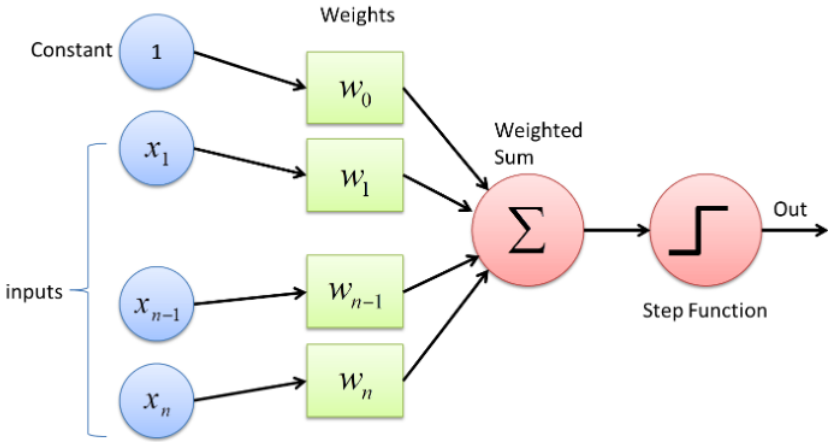


Figure 2.2: Graphical representation of a perceptron - a single layer network with one neuron.

2.7.2 The Forward Pass

The purpose of a neural network is to optimise its weights. In order for the network to be able to learn a mapping from the data it has been fed, the output of neurons is typically connected to the input of other nodes in subsequent layers. Each neuron calculates its weighted sum of inputs and applies an activation function to the sum to produce its output. This is called the Forward Pass.

The error between the predicted output and the target variable is computed using a loss (cost) function. The goal is to minimise this error in order to increase the accuracy of the network. One of the most popular loss functions is the Mean Squared Error (MSE). Given in Equation (2.12), MSE computes loss L, the difference between the target v and predicted output y for all training samples i.

$$L = \frac{1}{n}\sum_{i=1}^{n}(v_{(i)} - y_{(i)})^2 \tag{2.12}$$

A lot of deep neural networks are feedforward, meaning that they flow in only one direction, from input to output. The backpropagation algorithm can be used to perform gradient descent optimisation and update the weights on networks with more than one hidden layer.

2.7.3 Backpropagation

During the backward pass, the error is propagated through the network using the chain rule of differentiation to calculate the gradient from the loss function with respect to a parameter (i.e., weights) of the network. Error is not directly a function of a weight; hence, the weight update is expanded as in Equation (2.13), where y_j is the output value, z_j is the weighted input value, and w_{ij} is the weight of the *jth* neuron in the *ith* layer.

$$\Delta w_{ji} = -\mu \frac{\partial L}{\partial y_j} \frac{\partial y_j}{\partial z_j} \frac{\partial z_j}{\partial w_{ji}} \tag{2.13}$$

The weights are updated by performing gradient descent optimisation. In order to do this, let's consider each of these partial derivatives in turn. First, the derivative of the loss function for the *jth* neuron with respect to the output value is given in Equation (2.14).

$$\frac{\partial L}{\partial y_j} = -(v_i - y_i) \tag{2.14}$$

Secondly, the sigmoid function (Equation (2.17)) is chosen as the activation function of a neuron, so the model can solve problems that are non-linear in nature. The derivative of the activation function with respect to the weighted inputs can be determined as in the left Equation (2.15). The derivative of the weighted inputs with respect to a weight is given on the right side of Equation (2.15).

$$\frac{\partial y_j}{\partial z_j} = y_j(1 - y_j) \qquad \frac{\partial z_j}{\partial w_{ji}} = x_j \tag{2.15}$$

Now, by substituting the results of Equation (2.14) and Equation (2.15) back into our original equation, we get the weight update rule for our network listed in Equation (2.16). The learning rate hyperparameter μ specifies how quickly the network is learning.

$$\Delta w_{ji} = -\mu \frac{\partial L}{\partial w_{ji}} = \mu x_j \cdot \underbrace{y(v_j - y_j)(1 - y_j)}_{error\ derivative\ \delta_j} \tag{2.16}$$

Careful selection of the learning rate and activation function is required. The activation function maps the input to the output for a neuron in a non-linear environment. Choosing an unsuitable function can cause problems such as gradient vanishing or "dying neurons". Popular activation functions include the sigmoid function or Rectified Linear Unit (ReLU). Choosing a learning rate that is too small might cause prolonged training times, while large values could cause overfitting. Overfitting is one of the main concerns in Machine Learning. It happens when the model learns the training data too well, failing to produce good results on more general input sets.

The weight update rule is a generalisation of the Delta Rule [41] and can be reduced as shown in Equation (2.17), relying on the use of the aforementioned chain rule to compute gradients for each layer.

$$\Delta w_{ji} = -\mu x_j \delta_j \tag{2.17}$$

The forward pass, error computation, backward pass, and weight update steps are repeated in a process called an "epoch". The neural network will continue to train over epochs until the error converges to a minimum.

2.7.4 Long Short-Term Memory Networks

A Recurrent Neural Network (RNN) is a type of neural network that is specialised for processing sequential data [24]. Like regular neural networks, RNNs use training data to learn. What distinguishes them from the rest is their "memory", namely the ability to take information from prior inputs to influence the current input and output. This makes them suitable for processing sequential and temporal data.

Like feedforward and convolutional neural networks (CNNs), RNNs use training data to learn. They are distinguished by their "memory", as they take information from prior inputs to influence the current input and output. While traditional deep neural networks assume that inputs and outputs are independent of each other, the output of RNNs depends on the prior elements within the sequence.

RNNs train in a different way from fully connected neural networks by leveraging the backpropagation through time (BPTT) algorithm to determine the gradients. This is slightly different from traditional backpropagation in the sense that the forward activity has to be computed at each time t, then stored and used to backpropagate the loss through all the timesteps.

Long Short-Term Memory (LSTM) is a form of gated RNN designed to overcome the vanishing gradient problem. In 1991, Hochreiter identified that RNNs suffer from the vanishing gradient problem [20]. The issue occurs when the gradient used to update the parameters of the network becomes very small during backpropagation, making it difficult to train the network effectively. That is, if the previous state that is influencing the current prediction is not from the recent past, the RNN may not be able to accurately predict the current state. Introduced by Hochreiter and Schmidhuber in 1997, LSTMs address this issue by introducing a memory cell that allows the network to selectively remember or forget information from previous time steps [21]. The memory cell is controlled by three types of gates: input gates, output gates, and forget gates. A forget gate is reliant on a sigmoid function (Equation (2.17)), enabling the network to selectively update its memory cells and forget previous states, capturing long-term dependencies without vanishing gradients.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.17}$$

2.8 Summary

This chapter was an extended summary of the technical background that this project is reliant on, providing the reader with enough background to understand the scope and implementation of this project. It aims to offer insight into the algorithmic traders existing in the literature and how they operate. The test bed, TBSE, is detailed to provide evidence on the claim that asynchronous simulations better reflect real-life markets. Also, the extensive research performed on Deep Learning models offers motivation and an explanation of why and how our new trader will operate. The purpose is to create an LSTM-based agent and evaluate it against existing strategies using the right experimental methodologies.

Chapter 3

Project Execution

3.1 Integrating TBSE with DeepTrader

The core of this project relies on using TBSE, as introduced in Section 2.3. It was used to generate the large amounts of data required for training the LSTM network used by DeepTrader when running against the legacy trading strategies. TBSE is written in Python 3.4+, so for consistency and compatibility issues, most of the project was written using the same version of Python. The components that make the exception include the Dockerfile used for creating the image used for running the cloud clusters, the YAML job files used by Kubernetes to deploy the containers on AWS EC2 instances, and shell scripts concerning code checks and BC4 job execution. The core of the project, which includes TBSE and all the utilities used by DeepTrader, has been designed as a stand-alone application, with each sub-component being a Python package comprised of multiple modules. The Dockerfile included contains all the necessary information for packing the application along with its dependencies for portable deployment across different environments.

The system was designed to be modular and portable to suit collecting data for training and evaluating the model. The decision to isolate the simulation and model creation intends to ease reusability and help manage and track the heavy computation that they require. DeepTrader was integrated into TBSE as a plug-in component, implementing the existing interfaces that all the strategies use. Both DeepTrader and TBSE are separate packages operating inside a parent one, linked by a parent class that both the experiment and training environments implement. The project was treated as a real product, using GitHub for all development lifecycle purposes. GitHub actions have been defined to create pipelines for ensuring code quality compliance and dependency vulnerability checks. The most notable dependencies that we use are Tensorflow, Keras for Deep Learning and the Boto3 AWS SDK library. AWS S3 buckets were used for unified storage of all the data generated by the simulations. The Docker container repository was used as a remote location for the built image that the Kubernetes pods used to run the application container.

3.1.1 Market Simulation Price Ranges

TBSE was designed to use real-world historical data to introduce variability in its supply and demand schedules. Together with the option of setting the maximum and minimum price ranges for a customer order, the simulation enables simulating market sessions while avoiding repetition and bias. In our training and experimentation sessions, we used IBM stock price data from the August 31, 2017 NYSE trading day.

3.1.2 Feature Selection

DeepTrader was re-written to work on TBSE, but the original LSTM network architecture was preserved for consistency with the work of Cliff and Wray that demonstrated the performance of the model, inspiring this project. The data used as input to the model is generated by taking snapshots of the Level-2 LOB data, updated each time a trade occurs. The 13 multivariate input features that the DLNN uses along with its target variable are as follows:

1. The time t of the trade relative to the start of the market session, in seconds
2. The type of customer order type used to generate the quote initiating the trade is 1 for a "bid" order and 0 for an "ask" order.
3. The limit price of the customer order of the trader that initiated the trade
4. The midprice of the LOB at time t, as defined in Section 2.4
5. The microprice of the LOB at time t as defined in Section 2.4
6. The LOB imbalance at time t, as defined in Section 2.4
7. The spread of the LOB at time t as defined in Section 2.4
8. The best (highest priced) bid on the LOB at time t
9. The best (lowest priced) ask on the LOB at time t
10. The difference between the current time and the time of the previous trade
11. The total quantity of all quotes on the LOB at time t
12. An estimate P^* of the competitive equilibrium price as defined in Section 2.4.
13. Smith's α metric using the P^* estimate of the competitive equilibrium price at time t, described in Section 2.4,
14. The target variable: the price of the trade.

TBSE only produces output relating to the profits of the trading strategies of a specific market session. In order to accommodate the 14 features described, the code was adapted to generate one output CSV file per simulation containing snapshots of the Level-2 LOB data every time a trade occurs.

3.2 Neural Network Architecture

DeepTrader is based on a Long Short-Term Memory model, as first introduced in Section 2.7.4. Our LSTM is made up of three distinct hidden layers. The first layer is an LSTM with 10 units (neurones) that takes in a (13, 1)-shaped input. The next two are Dense layers, each with 5 and 3 units, respectively. All hidden layers use the Rectified Linear Unit (ReLU) activation function, one of the most commonly used activation functions in deep learning at the moment of this paper's writing. The equation for ReLU is defined in Equation (3.0). The last Dense layer of the model is the output layer, made up of one unit and a "linear" activation function suitable for a continuous output variable. The architecture used is illustrated in Figure 3.1.

$$ReLU(x) = \max(0, x) \tag{3.0}$$

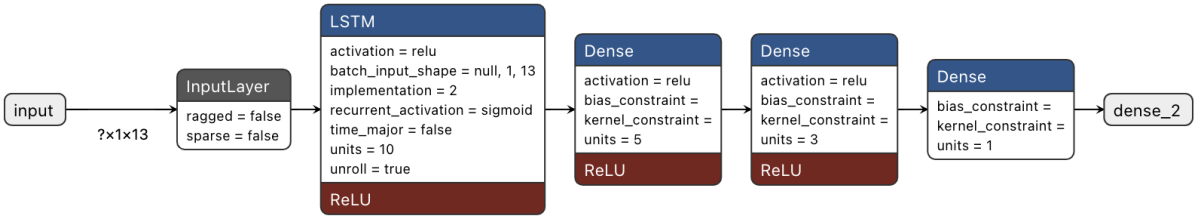


Figure 3.1: Architecture diagram of the DLNN architecture we are using, from input to output.

3.3 Producing the training data

TBSE provided five working trading agents that were used to generate the training data. In order to diversify the data and cover a lot of market scenarios, the simulations were run using different proportions of the trading strategies. TBSE recommends running market sessions with no more than 40 traders, otherwise risking unstable behaviour, so each market session had 40 traders. The following proportions of 20 traders per each side of the exchange (buyers or sellers) were used to generate the training data: (5, 5, 5, 5, 0), (8, 4, 4, 4, 0), (8, 8, 2, 2, 0), (10, 4, 4, 2, 0), (12, 4, 2, 2, 0), (14, 2, 2, 2, 0), (16, 2, 2, 0, 0), (16, 4, 0, 0, 0), (18, 2, 0, 0, 0), (20, 0, 0, 0, 0). Each number on a specific position corresponds to a population of traders of a certain type for a market simulation. For example, for the specification (12, 4, 2, 2, 0), there are 12 ZIC, 4 ZIP, 2 GDX, 2 AA, and no Giveaway traders for both the buyers and sellers sides.

All the unique permutations of each strategy were used so that TBSE ran an equal number of times across the simulations, summing to a total of 270 different trader schedules. Each schedule was executed for 44 individual trials, amounting to $270 \times 44 = 11880$ market sessions. Each session represented an hour of simulated market time, requiring a bit over one minute of real time. Running on a single computer, generating this amount of data would require approximately 8.6 days of continuous execution. Given the time constraint, the decision was made to use cloud computing.

In order to quickly set up the code to run in a fresh environment, we used Docker. Docker is a free tool that allows you to ship code alongside all its dependencies in a "container" which is a running instance of an "image". This way, the code can be deployed as an application on any host machine, as it will run on its native platform inside the container. We used the Docker Image Registry as a remote storage point for the most up-to-date version of the system. Provisioning lots of compute nodes requires a lot of time and is prone to errors. In order to manage the creation and provisioning of cloud resources, we used Kubernetes. Kubernetes (or K8s) is an open-source tool used to provision clusters of "pods", which contain the running containers. We used an AWS service called Elastic Kubernetes Service (EKS), which allows the user to create K8s clusters without the need to manually configure networking and the worker nodes. This way, by submitting a job manifest, we could spin up up to 9 instances (limited by the AWS Learner account) of AWS EC2 instances and execute our code with minimal effort. The job automatically instructs the compute nodes to pull the latest Docker image and run it as a container.

Generating these large amounts of files on multiple machines is hard to maintain, so TBSE was adapted to use the Python Boto3 (1.17.69) library to write to a cloud storage location, namely an S3 bucket. S3 buckets are offered by AWS as a low-cost, highly available remote storage system. This way, no matter from where the Docker container was run, the simulation output files were being written to the same location. To avoid overwriting and to help keep track of progress, each CSV file was named in the format `number_of_market_schedule-timestamp_of_execution.csv`. Figure 3.2 illustrates how the different cloud services are working to generate the training data.

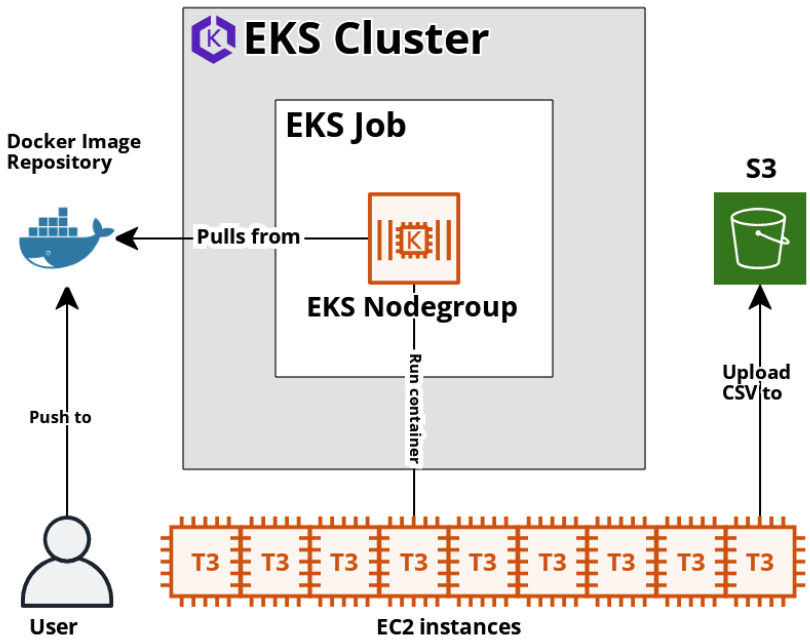


Figure 3.2: Architecture diagram of the cloud services used to generate our data.

3.4 Data Preprocessing and Curation

The files generated amount to roughly 13 million LOB snapshots, one per line. In order to save time and prepare the data for training, the Python Pickle library was used to serialise the CSV files to a large byte stream file.

It is generally good practise to normalise the inputs of a network due to performance concerns, particularly for Deep Learning architectures like LSTM, which our DLNN is based on. Normalising the inputs helps ensure that all features are contained within a similar range and prevents one feature from dominating the others. This is the case for our 13 input features and target variable that we described in Section 3.1.2. For example, the time runs to values up to 3600, while the type of order can only take values of 0 and 1. So by normalising, we would only have values in the [0,1] interval. By doing this, we improve the convergence of the optimisation algorithm and help the model generalise better to new data.

There are a number of normalisation strategies to choose from. Standardisation, min-max scaling, and power transformation are all suitable for a mix of continuous and binary variables, like the ones generated by TBSE. Given that we are working with multivariate features derived from financial data, it is important to preserve their scale while using an easy-to-understand model. That is, the choice was made to use min-max normalisation. Min-max normalisation scales each feature in the interval $[0, 1]$ by using its maximum and minimum values in the training data. Equation (3.0) is used to normalise the input features for each training sample, where x is the original value, x_{min} and x_{max} are the minimum and maximum values in the dataset, and x_{norm} is the normalised value.

$$x_{\text{norm}} = \frac{x - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}} \tag{3.0}$$

The data generated by the TBSE was examined for any irregularities or outliers. The only issues found were disproportionately large values for time and one of the other features. The trials containing these irregularities were discarded and rerun to preserve the consistency of the dataset. Contrary to the usual practises for training a DLNN, which consist of splitting the dataset into training, validation, and test subsets, we used all the dataset for training. The performance of the model was instead evaluated by how well DeepTrader performed against other traders in the Threaded-BSE in terms of profit per trader.

This is motivated by the nature of our target variable, the price of the order submitted by a trader (see Section 3.1.2). The only purpose of our model is to turn a profit, so testing order prices brings no benefit to our scope. Our dataset is large and was generated using stochastic simulations; thus, DeepTrader doesn't learn to replicate specific scenarios; rather, it grows its ability to adapt and generalise in any conditions.

3.5 Training the network

When dealing with large datasets, training should be done in batches in order to accommodate memory limitations and speed up training. Our network accommodates this by having a custom data generator that implements the Sequence class, which Keras uses to train a model in batches. To balance accuracy and training times, a batch size of 16384 was chosen. The learning rate of $\mu = 1.5 \times 10^{-5}$ was deemed the sensible choice, trying to balance potential overfitting and long convergence times, as already proven in previous implementations of DeepTrader. The DLNN uses the Adam optimizer for its ability to efficiently converge to a good solution, prevention of overfitting, and incorporation of momentum, which helps speed up learning and improve generalisation performance.

In order to speed up the training, the LSTM layer uses unrolling, which processes input sequences in parallel instead of sequentially. During training, 28 worker nodes were used to pre-load the batches in memory in order to reduce computational overhead. The network was trained using Blue Crystal 4, leveraging the power of its powerful GPUs, which Tensorflow is designed to optimise on. The training time required is approximately 22 hours.

The model was trained for 20 epochs. An epoch refers to a single pass of the entire dataset through the neural network. During each epoch, the model is exposed to each point of the dataset once. The training loss (cost) was calculated using the Mean Square Error (MSE) in Equation (2.12). After each epoch, the training loss dropped, with an especially hard fall during the first 4 passes. After that, the loss curve resembled a logarithmic scale, approaching 0 on the last epoch, as illustrated in Fig 3.3.

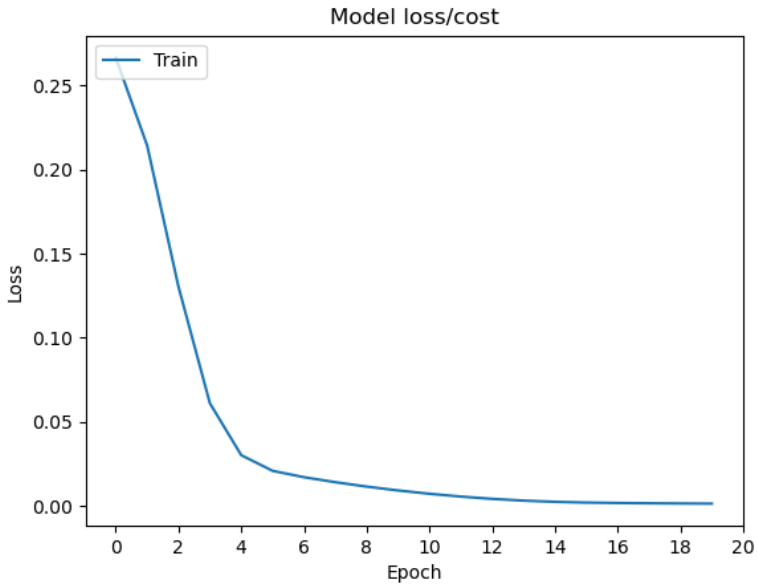


Figure 3.3: Loss curve curve calculated with the MSE after each epoch.

3.6 Adapting DeepTrader for High-Frequency Trading

Because TBSE introduces the idea of latency, the original code that DeepTrader used to interact with the market has been rewritten with the purpose of reducing computational overhead. The speed with which the trader generates market orders based on customer quotes can make or break a potential trade, regardless of how well the strategy adapts its price. In the TBSE, each trader operates as a separate thread, competing against the others for the best prices, so speed matters, just like in a real-world financial market. The code is written to maximise the use of the cache by loading static arrays in memory wherever possible and reducing repeated iterations on arrays. Also, it makes use of optimised Numpy methods to reduce the computational overhead generated by arithmetic operations. These changes may have a small impact when benchmarked in a sequential setting, but they have a considerable impact in a multi-threaded environment with thousands of calls to the same functions.

3.7 DeepTrader in TBSE

To be able to trade in the TBSE, DeepTrader must implement the Trader class, which all the other traders inherit from. This means that it uses the `get_order()` function, which has the purpose of getting the trader's order based on the current state of the market. In order to do this, DeepTrader uses information from the LOB to create input for its LSTM, generating a market order. As in training, the input has to be normalised to fit in the range $[0, 1]$ before the prediction and denormalized back to its human-readable format when used to generate a market order.

The main rule for a trader is that it cannot trade at a loss, buying or selling a commodity. It seldom happens that the LSTM will quote a price that, if used as is, would generate a loss for the customer placing the order. To prevent this, the price is checked before an order is submitted. But if the model fails to generate a good price, it doesn't mean that we should throw the order away and trade at the limit price, the same way the Giveaway trader does. Instead, DeepTrader will behave as a Shaver trader, taking the best bid or ask from the market and shaving a penny of its price. If that price does not match its limit, it will shave off a penny of its own limit price, trying to turn a profit even at its last resort. Listing 3.1 details the `get_order()` function of DeepTrader. The behaviour of DeepTrader and the limitations of its underlying DLNN will be presented in coming sections, along with an in-depth analysis of performance in one-to-many and balanced group tests.

```
def get_order(self, time, countdown, lob):
    """Implementation of get_order from Trader superclass in TBSE."""

    if len(self.orders) < 1:
        order = None
    else:
        coid = max(self.orders.keys())
        self.limit = self.orders[coid].price

        x = self.create_input(lob, otype=self.orders[coid].otype)
        normalized_input = (x - self.min_v_features) / (
            self.max_v_features - self.min_v_features
        )

        normalized_input = np.reshape(normalized_input, (1, 1, -1))
        normalized_output = self.model(normalized_input).numpy().item()

        denormalized_output = (
            (normalized_output) * (self.max_v_target - self.min_v_target)
        ) + self.min_v_target
        model_price = int(round(denormalized_output, 0))
```

```
        if self.orders[coid].otype == "Ask":
            if model_price < self.limit:
                model_price = self.limit + 1
                best_ask = lob["asks"]["best"] or TBSE_SYS_MIN_PRICE
                if self.limit < best_ask - 1:
                    model_price = best_ask - 1
        else:
            if model_price > self.limit:
                model_price = self.limit - 1
                best_bid = lob["bids"]["best"] or TBSE_SYS_MAX_PRICE
                if self.limit > best_bid + 1:
                    model_price = best_bid + 1

        order = Order(...)
        self.last_quote = order
    return order
```

Listing 3.1: `get_order()` function of DeepTrader

3.8 Experiments

A trader is designed and built with the purpose of making profit, so choosing profit per trader type as our performance metric was deemed the best choice. Section 2.5 introduces and details a set of methodologies used to compare trading strategies, arguing our choice for one-to-many and balanced group tests.

After running a market session, a strategy is said to win if it has a bigger profit per trader. Thus, we are also going to compare the proportion of wins for DeepTrader versus the other four types of traders: ZIC, ZIP, AA, and GDX. These 4 trading strategies were chosen as they are the most relevant in the literature, with AA, GDX, and ZIP being "super-human" traders, amongst the first to be proven to outperform humans.

We have a total of 8 head-to-head trading strategy comparisons, both one-to-many and balanced groups, each consisting of 500 market sessions with a total of 40 traders. The experiments were run using the same cloud infrastructure described in Section 3.3 using a dedicated S3 bucket for centralised storage.

3.8.1 One-to-many tests

The one-to-many tests were carried out to observe how DeepTrader performs as a "defector" on both the seller and buyer sides. TBSE is designed to shuffle around the positions of each trader side in order to reduce bias and create fair markets. In each of the 500 market sessions, the trader population was made up of 2 DeepTrader strategies and 38 other traders, equally distributed as buyers and sellers. The other market factors, such as session length and supply and demand schedules, were identical to the conditions used when generating the training data, as detailed in Section 3.3.

When running a very small number of traders of one type against a large population of traders of another strategy, it is sometimes the case that the traders might not trade at all. This is due to customer orders that cannot be fulfilled by a trader. For example, if the customer limit order price for buying a unit is $p = 50$ and the best ask on the LOB is $p_{best} = 80$, the market order placed by the trader will not cross the spread; therefore, another trader from the adversary strategy might "win" it first. On other occasions, the "defector" strategy might achieve disproportionately high profits, beating its competitor by a factor of up to 6. For this reason, to avoid an unfair comparison and having a very skewed dataset, the decision was made to exclude trials where one of the strategies scored a profit three times bigger than its competitor.

Plots (a) to (h) in Figure 3.3 display box plots of the distribution of profits. Box plots are a useful way to visualise the distribution of a dataset. For each plot, the box represents the interquartile range, the range between the first quartile and the third quartile. The line inside the box represents the median of the dataset. The whiskers represent the data within 1.5 times the interquartile range, with the diamond-shaped points outside them being considered outliers from a data distribution point of view.

The barplots (i) to (p) in Figure 3.4 display the percentage difference between the profits of DeepTrader and the adversary strategies. For example, if a bar is blue and sits at the 150% point on the y-axis, it means that DeepTrader was 1.5 times more profitable than the competitor; otherwise, a red bar sitting at -50% tells the reader that DeepTrader was half as profitable for that market simulation. It is designed to be an informative way to quantify and visualise the absolute performance difference between the strategies by only comparing proportions.

The plots (q) to (x) depicted in Figure 3.5 contain simple line plots depicting the profit per trader type in a specific experiment. The market trial pairs are sorted in ascending order by the adversary strategy's profit. The purpose of this is to display the data without the disturbance caused by the different seeds that were used in the stochastic elements of the simulation when running on a specific machine. Because the experiments were run in batches of 50 on different cloud nodes, the random functions in TBSE used different seeds, resulting in a variety of market conditions. These plots are a useful way to portray the qualitative results of the experiments, showing how DeepTrader sits relative to the curve produced by the ascending profits of its competitor. To improve the quality of the visualisation, the profit lines of DeepTrader have been smoothed using a moving average across eight trials.

3.8.2 Balanced group tests

The balanced group experiments (detailed in Section 2.5) were carried out in a similar way to the one-to-many, with the difference that we used 20 traders of a certain strategy on each side, for a total of 40 traders across all 500 trials. The market conditions are consistent with the trials that generated the training data. For consistency, the disproportionate profits were removed from the data, but in the case of comparing balanced groups of traders, these instances were very rare or non-existent in some experiments.

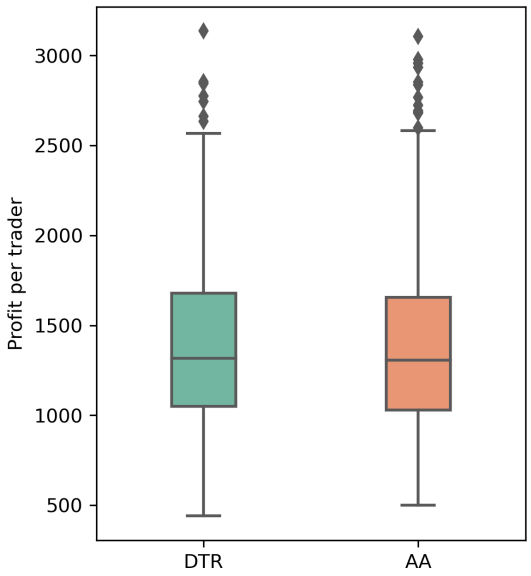
Same as in the one-to-many tests, the plots (a) to (h) in Figure 3.3 illustrate the distribution of profits across the trials; the barplots (i) to (p) in Figure 3.4 quantify the percentage difference between the competing strategies; and, lastly, the line plots (q) to (x) in Figure 3.5 portray their profit curves sorted in ascending order by the adversary's strategy values.

3.9 Counting head to head results

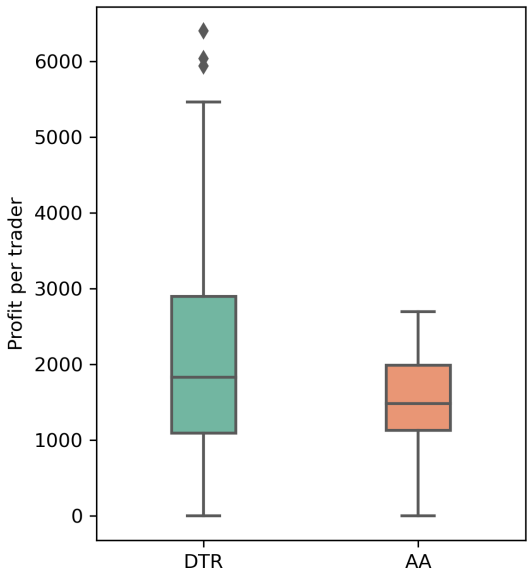
Figure 3.6 shows the difference between the number of wins of DeepTrader versus its competitors when counting individual trials. A "win" is said to be a market session where one strategy has more pure profit per trader than the other. We can see that DeepTrader has more head-to-head wins in almost all experiments. The critical analysis will determine if these results are consistent with the difference in profits, but they give good insight on the trend of results that DeepTrader manages to obtain.

3.10 Summary

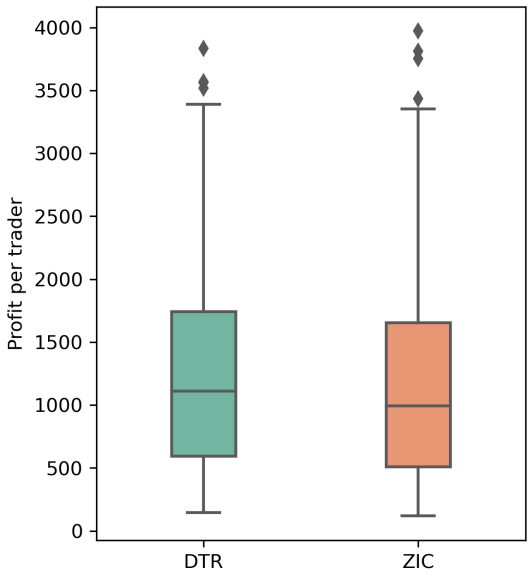
This chapter describes in great detail all the steps that have been taken to design, build, train, and evaluate DeepTrader to a high standard. The aim was to communicate and argue all the decisions taken in due process. The next chapter will provide a critical evaluation by performing qualitative and statistical quantitative analyses of the results, as well as discuss what they might mean for our project.



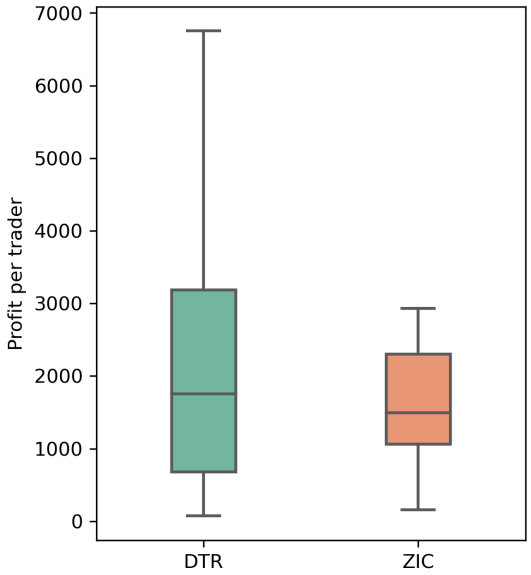
(a) Profit Distribution from the Balanced Group Tests between DeepTrader and AA.



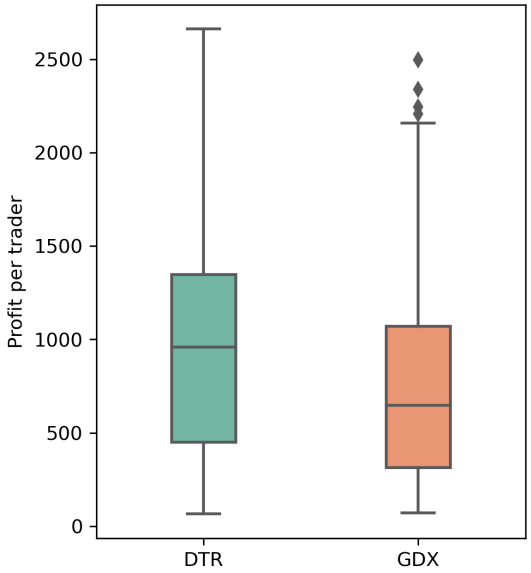
(b) Profit Distribution from the One-To-Many Tests between DeepTrader and AA.



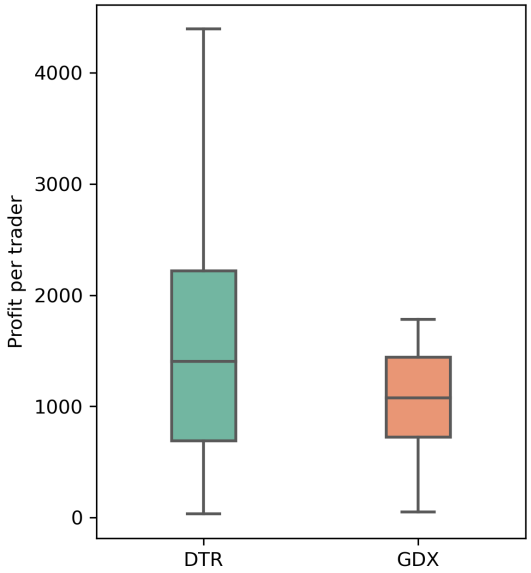
(c) Profit Distribution from the Balanced Group Tests between DeepTrader and ZIC.



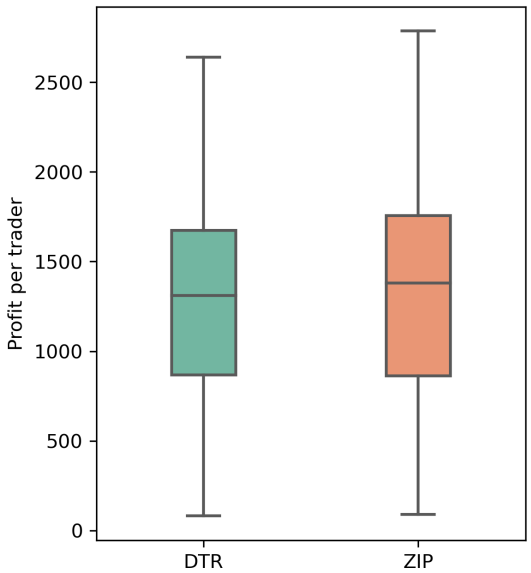
(d) Profit Distribution from the One-To-Many Tests between DeepTrader and ZIC.



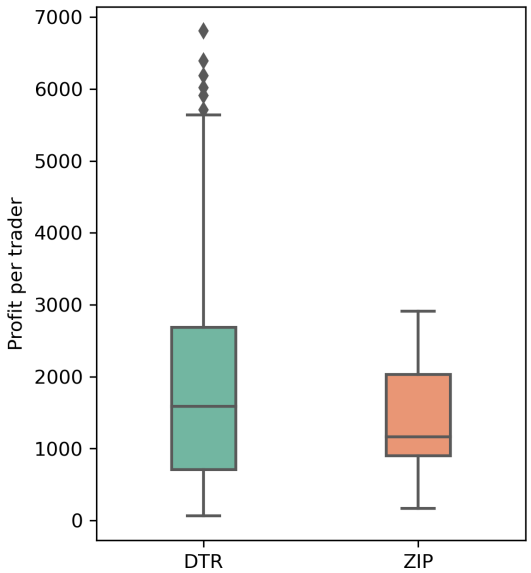
(e) Profit Distribution from the Balanced Group Tests between DeepTrader and GDX.



(f) Profit Distribution from the One-To-Many Tests between DeepTrader and GDX.

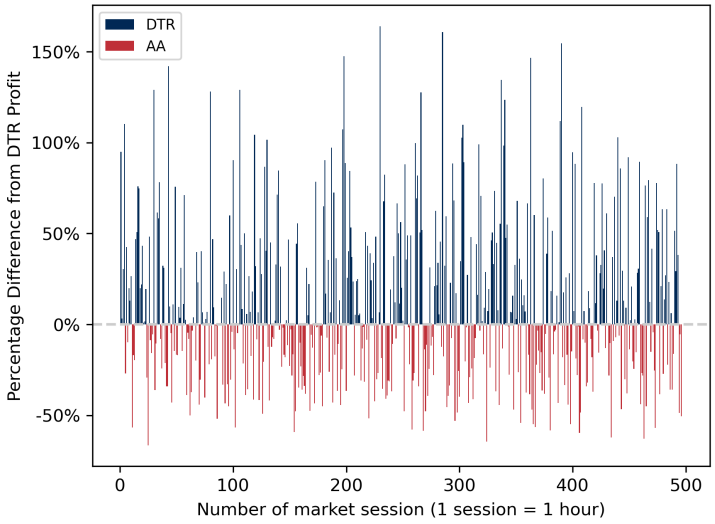


(g) Profit Distribution from the Balanced Group Tests between DeepTrader and ZIP.

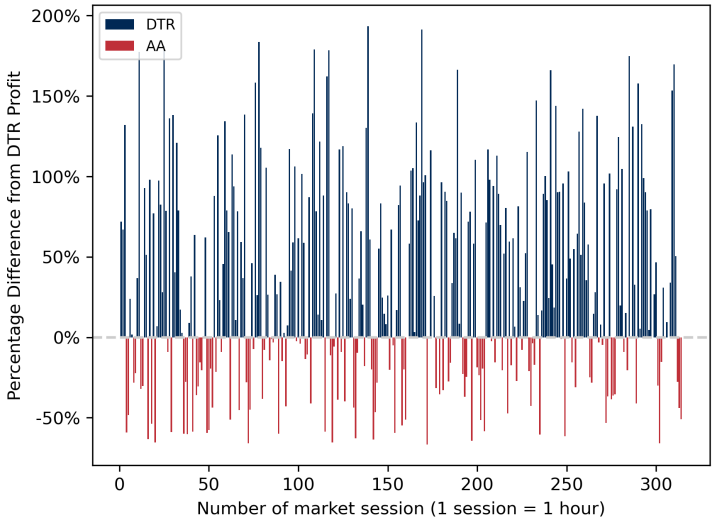


(h) Profit Distribution from the One-To-Many Tests between DeepTrader and ZIP.

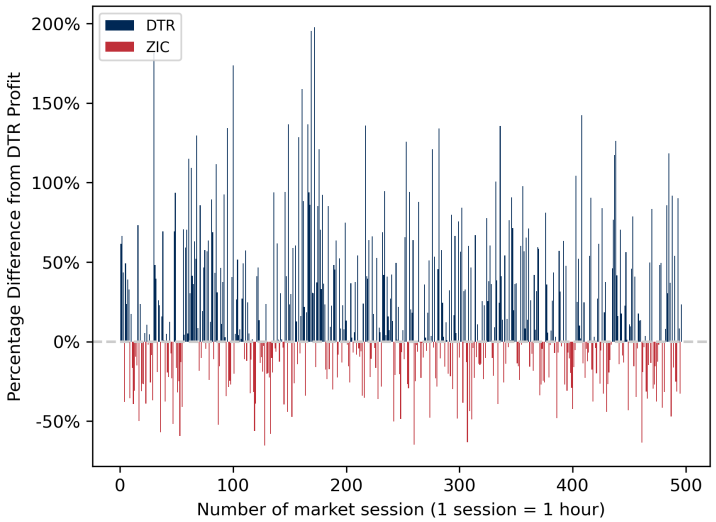
Figure 3.3: The subfigures display boxplots containing the distribution of profits from our experiments, excluding outliers.



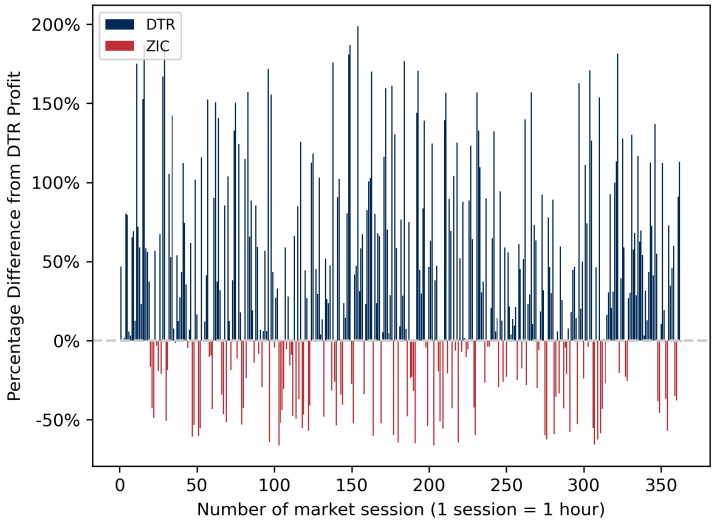
(i) Percentage profit difference of DeepTrader versus AA from the Balanced Group Tests.



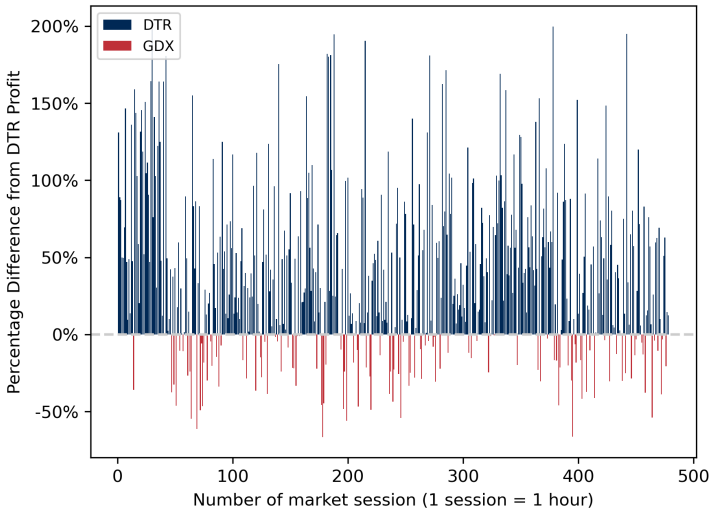
(j) Percentage profit difference of DeepTrader versus AA from the One-to-Many Tests.



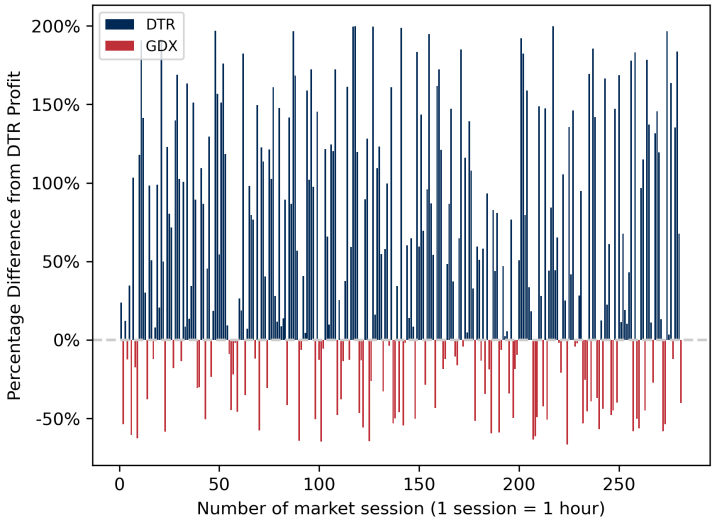
(k) Percentage profit difference of DeepTrader versus ZIC from the Balanced Group Tests.



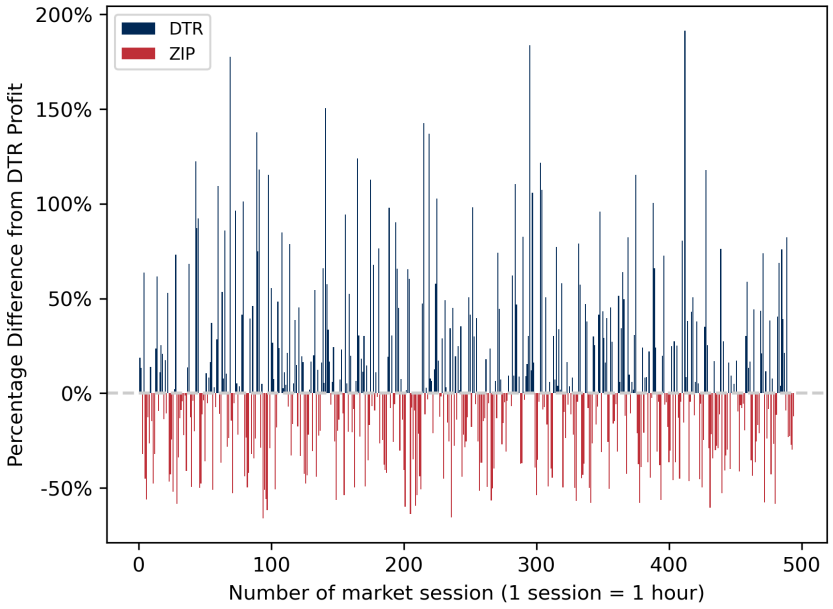
(l) Percentage profit difference of DeepTrader versus ZIC from the One-to-Many Tests.



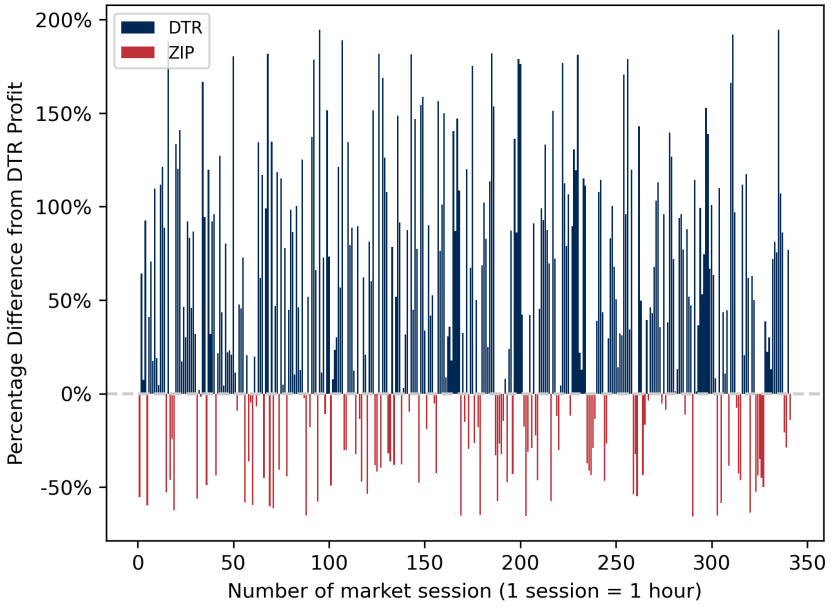
(m) Percentage profit difference of DeepTrader versus GDX from the Balanced Group Tests.



(n) Percentage profit difference of DeepTrader versus GDX from the One-to-Many Tests.

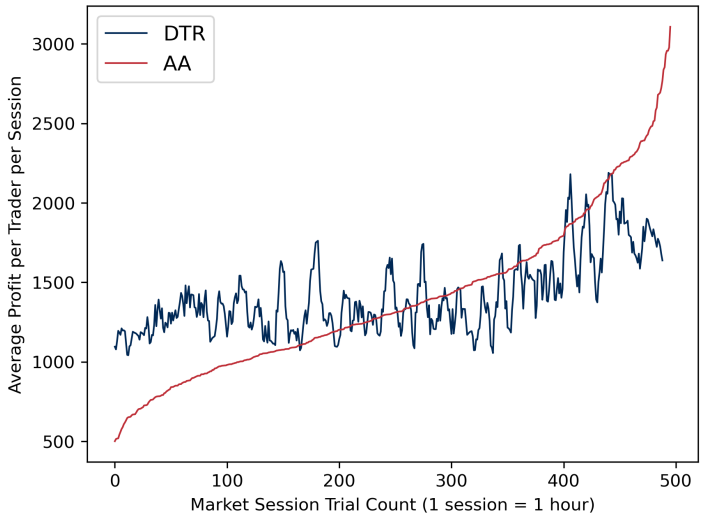


(o) Percentage profit difference of DeepTrader versus ZIP from the Balanced Group Tests.

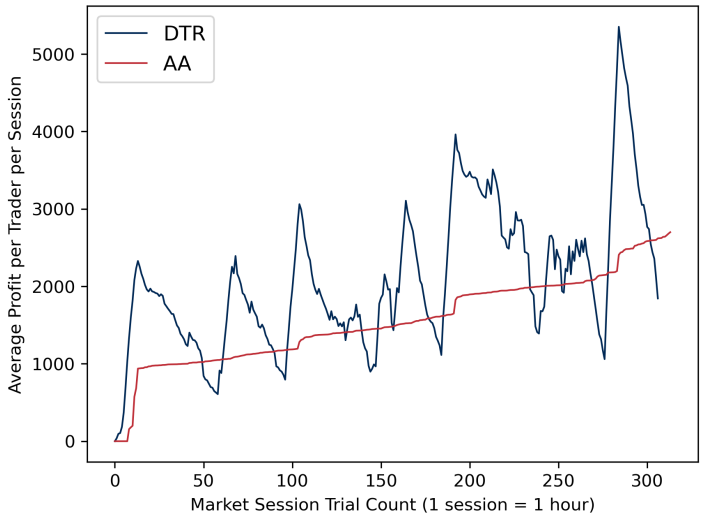


(p) Percentage profit difference of DeepTrader versus ZIP from the One-to-Many Tests.

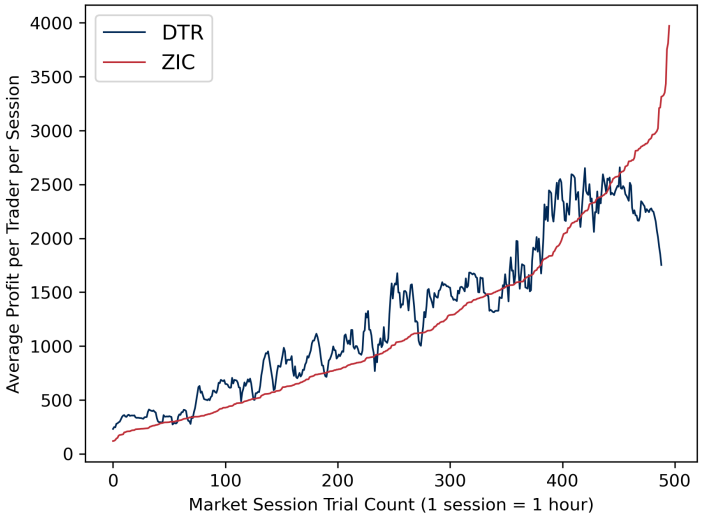
Figure 3.4: The subfigures display barplots containing the profit percentage difference between DeepTrader and other strategies, excluding outliers.



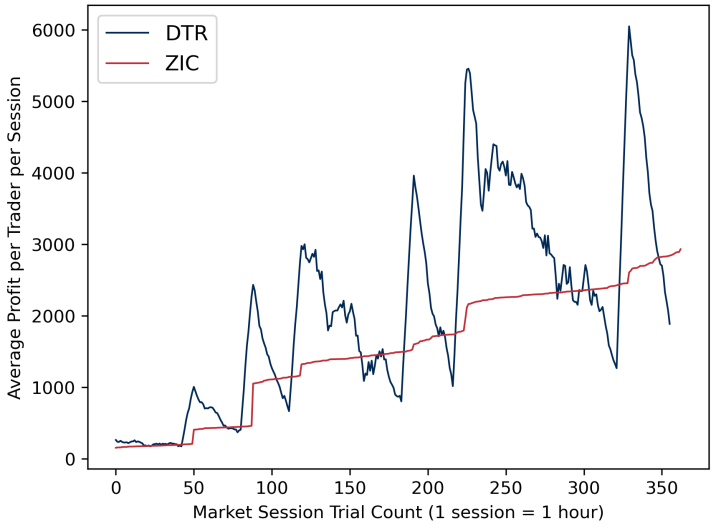
(a) Raw profit curves of DeepTrader versus sorted AA from the Balanced Group Tests.



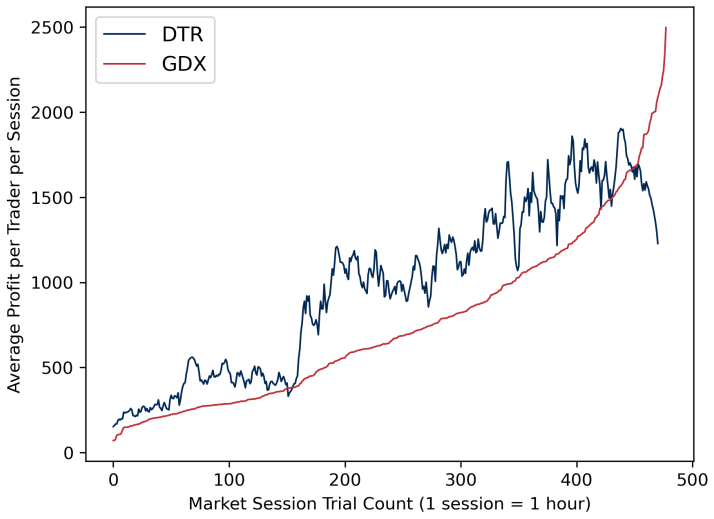
(b) Raw profit curves of DeepTrader versus sorted AA from the One-to-Many Tests.



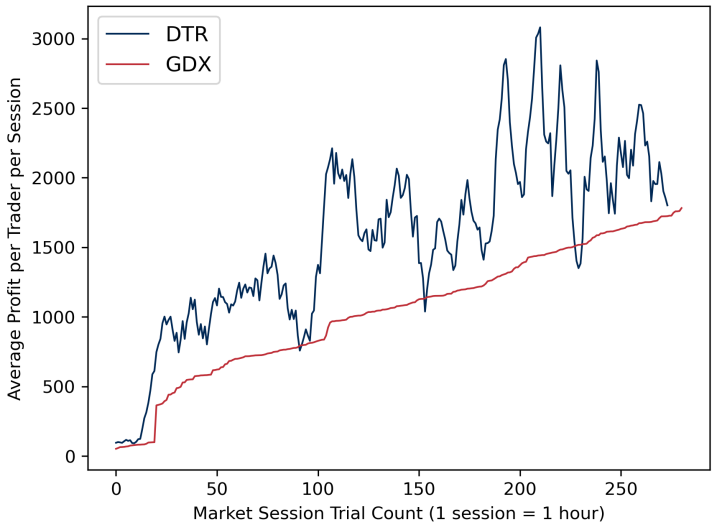
(c) Raw profit curves of DeepTrader versus sorted ZIC from the Balanced Group Tests.



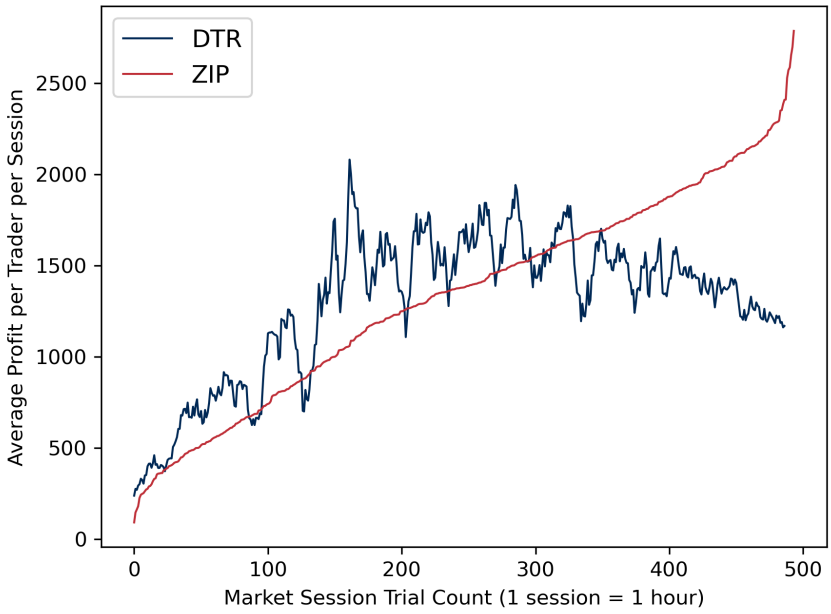
(d) Raw profit curves of DeepTrader versus sorted ZIC from the One-to-Many Tests.



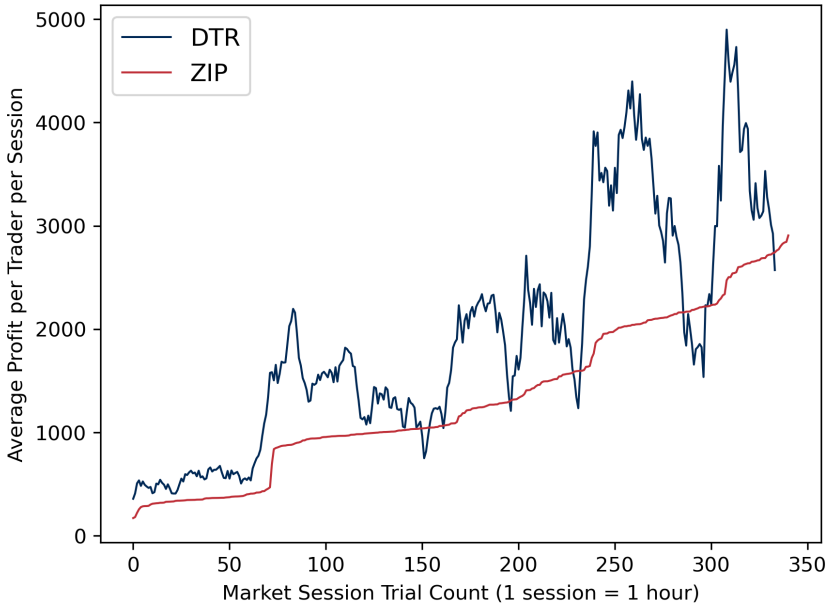
(e) Raw profit curves of DeepTrader versus sorted GDX from the Balanced Group Tests.



(f) Raw profit curves of DeepTrader versus sorted GDX from the One-to-Many Tests.



(g) Raw profit curves of DeepTrader versus sorted ZIP from the Balanced Group Tests.



(h) Raw profit curves of DeepTrader versus sorted ZIP from the One-to-Many Tests.

Figure 3.5: The subfigures display line plots containing the raw profits of DeepTrader versus competing strategies, sorted in ascending order by the adversary strategy's profits, excluding outliers.

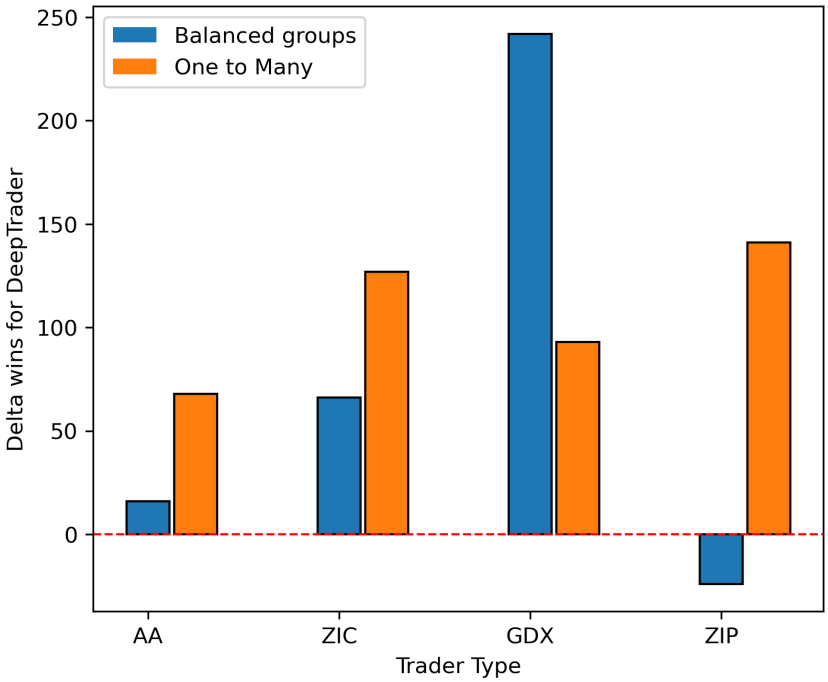


Figure 3.6: Displays the Δ_{wins} for DeepTrader versus the other strategies, with bars colored to represent different experiments.

Chapter 4

Critical Evaluation

This chapter is intended to provide extensive critical evaluation of the produced results in an objective manner. The results presented in Section 3.8 will be assessed in both qualitative and quantitative terms. The data produced in each experiment will be tested for normality and the relevant tests will be used to determine if there is any statistical meaningful difference in the performance of DeepTrader when compared to the other strategies in a variety of conditions.

Previous results of TBSE have challenged the status quo of the trading strategies hierarchy, previously agreed to be AA > GDX > ZIP > ZIC, by proving that in an asynchronous simulation this becomes ZIP > AA > ZIC > GDX. We will discuss which strategy is optimal from the results that are proven to be statistically significant. Also, we will come back to the aforementioned dominance hierarchy and check whether our results are consistent with it or not.

4.1 Testing for Normality

The Shapiro-Wilk test is a statistical test used to determine whether a given dataset is normally distributed or not. It is a goodness-of-fit test that assesses whether a sample of data comes from a population with a normal distribution. The null hypothesis of the test is that the data is normally distributed, and the alternative hypothesis is that the data is not normally distributed [43]. The formula for the Shapiro-Wilk test is given in Equation (4.0), where n is the sample size, $x_{(i)}$ is the ith order statistic (i.e., the ith smallest value) of the sample, $\bar{x}$ is the sample mean, and a_i are constants that depend on the sample size and the distribution being tested.

$$W = \frac{\left(\sum_{i=1}^{n} a_i x_{(i)}\right)^2}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \tag{4.0}$$

All the 16 cleaned datasets resulting from the 8 balanced group and one-to-many tests have been tested using the Shapiro-Wilk goodness-of-fit test, and each one of them has a resulting p-value lower than 0.05. Therefore, as this value is lower than the chosen significance level of $\alpha = 0.05$, we reject the null hypothesis and conclude that the datasets are not normally distributed.

The decision was made to process the datasets with the purpose of making them follow a normal distribution. For this scope, the datasets were normalised using "min-max" and "z-score" normalisation techniques. After re-running the Shapiro-Wilk tests, the resulting p-values increased but were still far from the $\alpha = 0.05$ significance level; therefore, we could conclude with confidence that the underlying distribution of the data does not follow a normal distribution.

As the pairs of profits for each experiment come from the same market session under the same conditions,

we can assume that they are related. A change in the behaviour of one type of strategy does affect the other and their subsequent profits.

By taking this matter and the normality results into consideration, on a sample size of maximum $n = 500$, the Wilcoxon signed-rank test was chosen to test whether the difference in profits is statistically significant or not.

4.2 The Wilcoxon signed-rank test

Created by Frank Wilcoxon in 1945, the Wilcoxon signed-rank test is a non-parametric statistical hypothesis test used to compare the locations of two populations using two paired (related) samples and does not make any assumptions about the underlying distribution of the data [27]. The null hypothesis of the Wilcoxon signed-rank test for two paired samples is that there is no difference between the median values of the two related samples. The alternative hypothesis of the test is that there is a significant difference between the median values of the two related samples. For all our tests, we have chosen a significance level of $\alpha = 0.05$. A p-value of the test higher than this threshold means that we cannot reject the null hypothesis, thus no winner can be identified, while a smaller value will accept the alternative hypothesis, concluding that one strategy outperforms the other.

The test is calculated using the formula given in Equation (4.0), where W is the test statistic, n is the number of pairs of observations, x_i and y_i are the ith observations in the two samples, $|x_i - y_i|$ is the absolute difference between the ith observations in the two samples, and $rank(|x_i - y_i|)$ is the rank of the absolute difference, calculated from 1 (smallest) to n (largest). The p-value is computed by comparing the observed test statistic W to the distribution of W values that would be expected under the null hypothesis of no difference between the two related samples.

$$W = \sum_{i=1}^{n} \operatorname{sgn}(x_i - y_i) \operatorname{rank}(|x_i - y_i|) \tag{4.0}$$

By looking at the distributions in Figure 3.3, we will notice that the standard deviation of the profit per trader is high for all experiments, indicating that the values are spread far from the mean. As mentioned in Section 3.8.1, this is caused by different seeds that the cloud machines use. Each market session was independent, but computers cannot produce perfect randomness [7]. This is the reason why 50 market sessions running on the same machine will have supply and demand curves that are stochastic but generated with the same seed, so they are closer to each other. For example, we can have sessions with both strategies achieving profits in the interval $[500, 1000]$ and some that might be in the interval $[1200, 2000]$, resulting in increased variability.

Additionally, the higher variance of DeepTrader in one-to-many experiments compared to its competitors is often justified by the fact that the profit per trader computed from 2 traders compared to 38 is naturally more variable. This happens because, when in the minority, traders are more exposed and can have very different performances depending on the order limits they are given and the market conditions they trade in. Meanwhile, a population of 38 traders will average out more constant profits as more traders can capture a wider range of circumstances.

The colours of the first row of the tables below indicate whether the experiment resulted in a win for DeepTrader (green), a loss for DeepTrader (red), or no statistical significance (grey).

4.3 Comparisons with AA

4.3.1 Balanced Group Comparison

Table 4.1 displays the mean $\bar{x}$, standard deviation σ, and p-value calculated using the Wilcoxon signed-rank test for profit-per-trader sample pairs resulting from the balanced group experiment between AA and DeepTrader. With a similar mean and variance and a test p-value higher than the 0.05 significance level, there is not enough statistical evidence to suggest there is a difference in performance between these two traders in a balanced group setting.

Trader	$\bar{x}$	σ	$p-value$
DeepTrader	1413	490	0.42
AA	1399	512	

Table 4.1: Displays results from the Balanced Group Tests with DeepTrader and AA.

4.3.2 One To Many Comparison

Table 4.2 displays the mean $\bar{x}$, standard deviation σ and p-value calculated using the Wilcoxon signed-rank test for profit-per-trader sample pairs resulting from the one-to-many experiment between AA and DeepTrader. With a considerably higher mean and a test p-value much lower than the 95% significance level, there is enough statistical evidence to suggest that DeepTrader outperforms AA in markets where it is the minority and AA the majority. However, the higher standard deviation suggests that the profits of DeepTrader are spread far from the mean. We conclude that DeepTrader is the higher-performing strategy in this experiment, but with increased variance.

Trader	$\bar{x}$	σ	$p-value$
DeepTrader	2062	1241	5.38×2^{-13}
AA	1555	564	

Table 4.2: Displays results from the One-to-Many Tests with DeepTrader and AA.

4.4 Comparisons with ZIC

4.4.1 Balanced Group Comparison

Table 4.3 displays the mean $\bar{x}$, standard deviation σ and p-value calculated using the Wilcoxon signed-rank test for profit-per-trader sample pairs resulting from the balanced group experiment between ZIC and DeepTrader. The means are close to each other and have a similar standard deviation, but DeepTrader has the upper hand. Thus, with a p-value smaller than the significance level of 95%, there is enough statistical evidence to suggest that DeepTrader is the dominant strategy in this type of experiment where ZIP and DeepTrader each make up 50% of the market participants.

Trader	$\bar{x}$	σ	$p-value$
DeepTrader	1268	826	0.0007
ZIC	1203	840	

Table 4.3: Displays results from the Balanced Group Tests with DeepTrader and ZIC.

4.4.2 One To Many Comparison

Table 4.4 displays the mean $\bar{x}$, standard deviation σ and p-value calculated using the Wilcoxon signed-rank test for profit-per-trader sample pairs resulting from the one-to-many experiment between ZIC and DeepTrader. DeepTrader has a mean 36% higher than ZIC but an 83% higher standard deviation. The low p-value strongly implies that there is enough statistical evidence to suggest that DeepTrader dominates ZIC in markets where DeepTrader is the minority and ZIC is the majority. The high standard deviation suggests an increased invariance of the profits obtained by DeepTrader. We therefore conclude that DeepTrader outperforms ZIC, but with a higher variance.

Trader	$\bar{x}$	σ	$p - value$
DeepTrader	2082	1597	1.26×2^{-14}
ZIC	1555	1530	

Table 4.4: Displays results from the One-to-Many Tests with DeepTrader and ZIC.

4.5 Comparisons with GDX

4.5.1 Balanced Group Comparison

Table 4.5 displays the mean $\bar{x}$, standard deviation σ and p-value calculated using the Wilcoxon signed-rank test for profit-per-trader sample pairs resulting from the balanced group experiment between GDX and DeepTrader. The higher mean points out that DeepTrader might be performing better than GDX, but the p-value very close to 0 strongly implies that there is enough statistical evidence to suggest that DeepTrader outperforms GDX in a setting where they both have the same number of traders.

Trader	$\bar{x}$	σ	$p - value$
DeepTrader	967	582	2.69×2^{-33}
GDX	753	502	

Table 4.5: Displays results from the Balanced Group Tests with DeepTrader and GDX.

4.5.2 One To Many Comparison

Table 4.6 displays the mean $\bar{x}$, standard deviation σ and p-value calculated using the Wilcoxon signed-rank test for profit-per-trader sample pairs resulting from the one-to-many experiment between GDX and DeepTrader. The DeepTrader mean profit in this case has a larger margin than in the balanced-group experiment, but with a standard deviation more than double that of GDX. The p-value smaller than the confidence level of 0.05 strongly suggests that there is enough statistical evidence to consider DeepTrader the better trader in this experiment. We conclude that DeepTrader outperforms GDX in a market where it forms the minority of the traders and GDX the majority, but with a higher variance.

Trader	$\bar{x}$	σ	$p - value$
DeepTrader	1539	1032	3.09×2^{-15}
GDX	1042	457	

Table 4.6: Displays results from the One-to-Many Tests with DeepTrader and GDX.

4.6 Comparisons with ZIP

4.6.1 Balanced Group Comparison

Table 4.7 displays the mean $\bar{x}$, standard deviation σ and p-value calculated using the Wilcoxon signed-rank test for profit-per-trader sample pairs resulting from the balanced group experiment between ZIP and DeepTrader. ZIP has a slightly bigger mean profit and a similar standard deviation to DeepTrader. Therefore, with a test p-value slightly under the significance level of $\alpha = 0.05$, there is enough statistical evidence to conclude that ZIP outperforms DeepTrader in this balanced group experiment.

Trader	$\bar{x}$	σ	$p - value$
DeepTrader	1277	542	0.043
ZIP	1333	573	

Table 4.7: Displays results from the Balanced Group Tests with DeepTrader and ZIP.

4.6.2 One To Many Comparison

Table 4.8 displays the mean $\bar{x}$, standard deviation σ and p-value calculated using the Wilcoxon signed-rank test for profit-per-trader sample pairs resulting from the one-to-many experiment between ZIP and DeepTrader. The mean profit of DeepTrader is sensibly higher than that of ZIP, but profits are more dispersed from the mean. With a test p-value close to 0, the Wilcoxon signed-rank test strongly suggests that the difference in profits between DeepTrader and ZIP is statistically significant in favour of DeepTrader. We therefore conclude that DeepTrader outperforms ZIP when it is the defector of a trader population, but with increased variance.

Trader	$\bar{x}$	σ	$p - value$
DeepTrader	1902	1413	1.95×2^{-19}
ZIP	1323	737	

Table 4.8: Displays results from the One-to-Many Tests with DeepTrader and ZIP.

4.7 Head-to-Head Comparison

So far, we have compared the mean profits from the two types of experiments we executed. Additionally, we are going to take a look at the head-to-head results. A trading strategy is said to "win" in a trial if it produces more profit per trader than its adversary. Let's consider the difference in wins in an experiment as Δ_{wins}. If we compare, for example, DeepTrader vs. AA, if $\Delta_{wins} > 0$, then DeepTrader outperforms AA in the experiment; otherwise, if $\Delta_{wins} < 0$, DeepTrader is outperformed by AA. Table 4.9 shows the head-to-head results from 500 trials per experiment, excluding outliers.

The results show near total dominance of DeepTrader, with the exception of the balanced group experiment with ZIP.

4.8 Summary of Results

The summary of all our results is presented in Tables 4.9, 4.10 and 4.11 for the head-to-head win count and each experiment type. They contain the outcomes of all the comparisons between ZIC, ZIP, AA, GDX, and DeepTrader. As the tables show, DeepTrader was only outperformed by ZIP in the balanced group tests and the win count. Also, no statistical evidence was found that it outperformed AA in the

Trader & Experiment Type	Δ_{wins}	% of Wins	Ratio of Wins
AA - Balanced Groups	16	52:48	1.04
ZIC - Balanced Groups	66	57:43	1.33
GDX - Balanced Groups	242	75:25	3.00
ZIP - Balanced Groups	−24	48:52	0.92
AA - One-to-Many	68	61:39	1.56
ZIC - One-to-Many	127	67:33	2.03
GDX - One-to-Many	93	67:33	2.03
ZIP - One-to-Many	141	71:29	2.45

Table 4.9: Displays all the head-to-head results of DeepTrader vs. other strategies, including the Δ_{wins}, % and ratio of wins over the 500 trials per experiment, excluding outliers. The red row represents a loss, the grey row has no statistical significance and the green rows represent wins of DeepTrader.

balanced group test, but it achieved more round wins and a higher mean profit. Put together, these results are truly impressive, as DeepTrader has outperformed in all of the other experiments and win counts.

The only mention is that in the one-to-many tests, DeepTrader outperformed but with a higher variability of profits over the course of almost 2000 market sessions, excluding the outlier cases, as mentioned in Section 3.8, where one of the strategies had a profit more than three times higher than the other. This is visually noticeable in Figure 3.3. The plots in Figures 3.4 and 3.5 also offer visual insight on how and in what way DeepTrader manages to obtain profit. Thus, DeepTrader is not guaranteed to be very reliable when run in just a few market sessions against the majority of other trading strategies.

In the beginning of this chapter, we mentioned the new dominance hierarchy presented in the paper that introduces TBSE. Because of some ambiguous results, such as the tests with ZIP, where in one type of experiment DeepTrader outperforms and in the other it is outperformed, we cannot draw a clear line as to where DeepTrader sits in that new hierarchy. Instead, we can make a new one by taking into account the degree to which DeepTrader dominated (or not) the four strategies.

So, ZIP performed the strongest against DeepTrader, followed by AA, followed by ZIC, and ending with GDX, which was the weakest contender against DeepTrader. Very interestingly, this is consistent with the hierarchy of TBSE, which is ZIP > AA > ZIC > GDX. This is proof that our implementation of DeepTrader is robust and performs based on the structure of the market, not stochastic events.

	Result	Comment
AA	Loss	Only experiment where DeepTrader lost
ZIC	Win	DeepTrader outperforms
GDX	Win	DeepTrader significantly outperforms
ZIP	-	No statistical difference in performance was recorded

Table 4.10: Displays a summary of results for the Balanced Groups Experiments. The red row represents a loss, the grey row has no statistical significance and the green rows represent wins of DeepTrader.

	Result	Comment
AA	Win	DeepTrader significantly outperforms with increased variance
ZIC	Win	DeepTrader outperforms
GDX	Win	DeepTrader significantly outperforms with increased variance
ZIP	Win	DeepTrader significantly outperforms with increased variance

Table 4.11: Displays a summary of results for the One-to-Many Experiments. The green rows represent wins of DeepTrader.

4.9 Real World Application

4.9.1 Results

The experiments described in Section 3.8 were designed to produce empirical results for comparing trading strategies in a fair and reproducible way, but they do not fully reflect real-life financial exchanges. Firstly, a real market is populated by hundreds or thousands of traders, each trying to maximise their own profits. The strategies that make money are not public, so it is highly unlikely that an entire market would be populated by only two trading strategies. Secondly, the distribution and number of traders in the buyer and seller groups may be constantly changing and unequal, unlike what was assumed for the scope of our project. And lastly, TBSE only accepts orders with a quantity of one, but in real exchanges, the quantity of a customer order influences the strategy of a trader.

Considering all these aspects, the results on the performance of DeepTrader against public-domain traders are impressive and promising, but they should only provide a starting point for researching how DLNN-based trading agents behave in real-life asynchronous financial exchanges.

4.9.2 Real High Frequency Trading

The DLNN architecture of DeepTrader is designed to be simple but effective. Models with a lightweight design can perform inference on new data faster, making them suitable for high-frequency trading. Real-world traders compete against each other at sub-second rates, so speed can mean more than accuracy. DeepTrader relies on relatively simple input and was designed to be as fast as possible, but in a real market, an agent might require more complicated data that would slow it down. Considering these facts, it can be concluded that DeepTrader does not necessarily reflect how an AI strategy might operate in the real world, but being implemented in TBSE, it offers a good proof-of-concept for such a model performing high-frequency trading.

4.9.3 Availability of Limit Prices

Out of all the LOB features outlined in Section 3.1.2 that DeepTrader is reliant on, perhaps the most difficult to obtain in a real-world setting is a trader's limit price. The other information is publicly available on the LOB of an exchange, but a trader disclosing its limit prices can compromise its competitive advantage in a market. This information can then be used by others to perform reverse engineering, disclosing the money-making strategy of the trader.

If we wanted to translate this project into the real world, we would need to "sit behind the desk" of a successful actual trader, collecting training data so that DeepTrader would be able to imitate its behaviour. That process would, of course, be automated, and after collecting millions of data points, DeepTrader could be trained to trade a certain financial asset. If one wanted to extend it to multiple assets, they would need to clone it and train it with the relevant data for each type of item they wanted it to trade.

Chapter 5

Conclusion

5.1 Summary of Outcomes

The main achievement of this project was the creation of an automated trading agent based on a Deep Learning Neural Network that performs very well against existing public-domain trading strategies in an asynchronous simulation, the TBSE, capturing dynamics present in real-world markets.

Referring back to the project approach first outlined in Section 1.5 , the main objectives and, now, achievements were:

1. Perform extensive research on existing literature on Artificial Intelligence in finance, with a focus on automated trading agents.
2. Build on the knowledge accumulated in the past 4 years of the degree to understand, plan, and design every bit needed to assemble this project.
3. Integrate and extend existing code to create a working platform for generating training data and running it in the cloud.
4. Create an automated pipeline for querying, cleaning, and compressing data stored in the cloud in a format ready for training.
5. Train, tune, and improve a profitable new iteration of DeepTrader, optimised for latency-sensitive markets, and test it against existing strategies.
6. Perform a critical evaluation of the performance of the model against existing agents using different methodologies and test for normality and statistical significance.

5.2 Project Status

The project has been completed to its full extent, creating a robust and scalable system to collect data for, train, and test a fast, high-performing AI trader. The system proved to win in six out of the eight experiments performed, produced similar profits, and lost in the remaining two. These are nevertheless impressive results.

In regards to the first aim, Chapter 1 outlines the detailed research and survey of the literature that were performed in the beginning stages of the project, starting with the origins of experimental economics and finishing with the state-of-the-art AI systems in modern finance. Section 1.1 clearly references the past work our project is building on.

Chapter 2 provides all the inner technical workings that are the pillar stones of this project, as proposed in the second objective. It treated in great detail the basics of LOB-based markets. If offered a comprehensive discussion of how TBSE works, how to apply different experiment frameworks, and the in-depth details of what makes models like DeepTrader work.

The third and fourth objectives were tackled and outlined in Chapter 2, providing an extensive walk-through of the technical aspects that have been executed - from adapting an existing codebase (Section 3.1) to building a system and that can successfully operate and replicate in a cloud environment (Section 3.3), as part of a pipeline for creating the training dataset for our model (Section 3.4). The steps involving training and tuning the model, as claimed by objective 5, were described in detail in Section 3.5. The tests performed and the methodologies chosen are described in Section 3.8.

Lastly, Chapter 4 presents the analysis performed on the results of our DeepTrader. The critical evaluation demonstrates that DeepTrader is making profit and outperforming other agents in most of the experiments, but there are issues regarding high variability that need further attention. Also, it is unknown why there was no significant difference when comparing with AA in a balanced group setting.

Going back to the research hypothesis stated in the abstract:

> Is a new high-frequency trader based on a "black-box" Deep Learning model, trained solely on observed behaviour from the LOB of a market capable of competing with and outperforming existing trading agents in an asynchronous, multi-threaded market simulation?

Reflecting on this, our version of DeepTrader has been shown to outperform most of and even completely dominate some existing strategies in a stochastic parallel market simulation. However, while interesting and impressive, we cannot state whether DeepTrader is superior or not and come to a definitive answer. Thus, further work can help build confidence in DeepTrader and bring us closer to drawing that line.

5.3 Future Work

5.3.1 Further Evaluation

There is essentially no limit on how much testing can be performed to test a certain hypothesis, but there is certainly an amount of testing that can get closer to a conclusion on it. Running more, longer market sessions in more varied conditions, emulating what happens in the real world, would bring more insight on the performance patterns of DeepTrader, answering where, when, and how often it outperforms the other strategies. Also, performing speed benchmarks to measure how code latency impacts profits would help explain the different behaviours of DeepTrader and its competitors. Shuffling trader schedules and ratios up to their exhaustion would narrow the gap between our results and the truth. Having more available resources could help leverage the full power that the cloud can bring and scale the testing framework to cover everything we mentioned in manageable timeframes, but beyond the scope of this project.

5.3.2 Using a Truly Realistic Simulation

TBSE has questioned the status quo of trading agents, opening up a new world of research possibilities by offering the possibility to run market sessions on a multi-threaded platform, emulating how thousands of traders compete with each other in a real financial exchange, for example, the New York Stock Exchange (NYSE). But it is really difficult to fully recreate what is going on in real life. There is a whole world of complexities, such as external market factors, trader sentiment, the fluctuating number of traders, the influence of large transactions, and many more, that are very difficult to capture in a simulation. There might be such software developed by financial institutions, but it would be expected to be closed-source and only used for tuning systems worth hundreds of millions of dollars.

If DeepTrader were to be trained on data generated from and tested in such a system, it would help reveal its vulnerabilities when performing in different scenarios. Perhaps the oversimplification of TBSE allows DeepTrader to be a competitive trader by reducing the dimension of data that it has to learn from. This technique of "throwing data" at such a simple model and hoping that it trains and performs well might have been a successful bet in our research, but when faced with an exponentially larger and more complex problem, the network might struggle to capture patterns and make profitable decisions. It would be very exciting to see DeepTrader perform in such scenarios, but beyond the available resources that an academic institution might possess on its own.

5.3.3 Training Data

The performance of a model depends on the reliability of its training data. Of course, the size of the input dataset is very important, but it would serve little purpose if the data it represents is of low quality. An effort has been made in this project to generate comprehensive trading data using the strategies implemented by TBSE. But the limitation lies in the relatively short time frame that was available for this purpose. Using a much larger and perhaps broader dataset can improve the performance and resilience of DeepTrader.

The strategies we used to build the training set contained known less profitable strategies: the one that recently proved to be underperforming in TBSE, GDX, and the "charity" Giveaway. One might argue that using these strategies might skew the dataset to be less reliable, which may be true, but also bring more diversity and realness to the trading conditions, as it might happen in real markets. Studying how DeepTrader performs when learning from more traders or from specific high-performing strategies would help define its behaviour better.

5.3.4 Feature Ablation Study

One question that a prospective reader of this piece of work might be pressed by at this point is: How and why exactly is DeepTrader, a simple neural network, performing so well? We have tried to give an explanation for that throughout this paper, but the full answer is a little bit more complicated. While we can definitively say that DeepTrader learns to map inputs to an output value and, by doing so in a fast manner, makes profit, we cannot provide insights from "under the hood". Chapter 2 makes an attempt at explaining the technicalities of this project, including how backpropagation and LSTMs work, arguing that this kind of network is well suited for our type of input data, but doesn't explain how DeepTrader "judges" LOB data to give a price to trade with.

To aid with this matter, a feature ablation study could be performed to determine what features matter the most and to what extent. One by one, input features could be "turned off", measuring the impact they have on the performance of our strategy. The insight provided would help us get closer to the answer of the question.

5.4 Final Words

This project has been intended to explore the use of Artificial Intelligence in creating a high-performing trading strategy in an asynchronous market simulation, opening up a new path for future research in this area. The emergence of AI not only in finance but in every aspect of human activity represents a turning point in the evolution of human civilization, perhaps equivalent to the invention of the internet or even the industrial revolution. We should be aware of the inherent risks, but most likely, the next few years will bring lots of exciting new technologies.

Fin.

Bibliography

[1] davecliff . Github - davecliff/BristolStockExchange: Bse is a simple minimal simulation of a limit order book financial exchange. https://github.com/davecliff/BristolStockExchange, oct 17 2022.

[2] Amira Hassan Abed. Deep Learning Techniques For Improving Breast Cancer Detection And Diagnosis. *International Journal of Advanced Networking and Applications*, 13(06):5197–5214, 2022.

[3] Md Zahangir Alom, Tarek M. Taha, Chris Yakopcic, Stefan Westberg, Paheding Sidike, Mst Shamima Nasrin, Mahmudul Hasan, Brian C. Van Essen, Abdul A. S. Awwal, and Vijayan K. Asari. A State-of-the-Art Survey on Deep Learning Theory and Architectures. *Electronics*, 8(3):292, mar 5 2019.

[4] Acumen Research and Consulting. Algorithmic Trading Market Size Expanding at 12.9Set to Reach USD 41.9 Billion By 2030. https://www.globenewswire.com/news-release/2023/01/31/2599023/0/en/Algorithmic-Trading-Market-Size-Expanding-at-12-9-CAGR-Set-to-Reach-USD-41-9-Billion-By-2030.html, jan 31 2023.

[5] Marco Avellaneda and Sasha Stoikov. High-frequency trading in a limit order book. *Quantitative Finance*, 8(3):217–224, 4 2008.

[6] Robert L Axtell and J Doyne Farmer. Agent-based modeling in economics and finance: past, present, and future. *Journal of Economic Literature*, 2022.

[7] James Bridle. The Crucial Thing a Computer Can't Do. https://slate.com/technology/2022/06/bridle-ways-of-being-excerpt-computer-randomness.html, jun 21 2022.

[8] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[9] Arthur le Calvez and Dave Cliff. Deep Learning can Replicate Adaptive Traders in a Limit-Order-Book Financial Market. *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*, 11 2018.

[10] Charles Cao, Oliver Hansch, and Xiaoxin Wang. The information content of an open limit-order book. *Journal of Futures Markets*, 29(1):16–41, 1 2009.

[11] Dave Cliff. Minimal-intelligence agents for bargaining behaviors in market-based environments. *Hewlett-Packard Labs Technical Reports*, 1997.

[12] Dave Cliff. An open-source limit-order-book exchange for teaching and research. In *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1853–1860. IEEE, 2018.

[13] Pádraig Cunningham, Matthieu Cord, and Sarah Jane Delany. Supervised Learning. *Machine Learning Techniques for Multimedia*, pages 21–49, 2008.

[14] Marco De Luca and Dave Cliff. Human-agent auction interactions: Adaptive-aggressive agents dominate. In *Twenty-second international joint conference on artificial intelligence*, 2011.

[15] Srividhya E, Jayanthi S, Suja Cherukullapurath Mana, V. R. Niveditha, and Amandeep Singh K. An Approach To Deep Learning. -, nov 30 2022.

[16] E. Fehr, G. Kirchsteiger, and A. Riedl. Does Fairness Prevent Market Clearing? An Experimental Investigation. *The Quarterly Journal of Economics*, 108(2):437–459, may 1 1993.

[17] Erik Gawehn, Jan A. Hiss, and Gisbert Schneider. Deep Learning in Drug Discovery. *Molecular Informatics*, 35(1):3–14, dec 30 2015.

[18] Steven Gjerstad and John Dickhaut. Price Formation in Double Auctions. *Games and Economic Behavior*, 22(1):1–29, 1 1998.

[19] Dhananjay K. Gode and Shyam Sunder. Allocative Efficiency of Markets with Zero-Intelligence Traders: Market as a Partial Substitute for Individual Rationality. *Journal of Political Economy*, 101(1):119–137, 2 1993.

[20] Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 91(1), 1991.

[21] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[22] investopedia. What Is a Trader, and What Do Traders Do? https://www.investopedia.com/terms/t/trader.asp, mar 15 2023.

[23] Suhwan Ji, Jongmin Kim, and Hyeonseung Im. A comparative study of bitcoin price prediction using deep learning. *Mathematics*, 7(10):898, 2019.

[24] Yijia Liu, Wanxiang Che, Bing Qin, and Ting Liu. Exploring segment representations for neural semi-markov conditional random fields. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 28:813–824, 2020.

[25] David Lucking-Reiley. Using field experiments to test equivalence between auction formats: Magic on the internet. *American Economic Review*, 89(5):1063–1080, 1999.

[26] An Njegovanović. Artificial intelligence: Financial trading and neurology of decision. -, 2018.

[27] J Petersson, TE Gordh, P Hartvig, and L Wiklund. A double-blind trial of the analgesic properties of physostigmine in postoperative patients. *Acta anaesthesiologica scandinavica*, 30(4):283–288, 1986.

[28] Michael Rollins and Dave Cliff. Which trading agent is best? using a threaded parallel simulation of a financial market changes the pecking-order, 2020. arXiv:2009.06905.

[29] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[30] Deekshith Shetty*, Harshavardhan C.A, M Jayanth Varma, Shrishail Navi, and Mohammed Riyaz Ahmed. Diving Deep into Deep Learning: History, Evolution, Types and Applications. *International Journal of Innovative Technology and Exploring Engineering*, 9(3):2835–2846, jan 30 2020.

[31] Thalita R Silva, Audeliano W Li, and Edson O Pamplona. Automated trading system for stock index using lstm neural networks and risk management. In *2020 international joint conference on neural networks (IJCNN)*, pages 1–8. IEEE, 2020.

[32] Justin Sirignano and Rama Cont. Universal features of price formation in financial markets: perspectives from deep learning. *Quantitative Finance*, 19(9):1449–1459, jul 9 2019.

[33] Vernon L. Smith. An Experimental Study of Competitive Market Behavior. *Journal of Political Economy*, 70(2):111–137, 4 1962.

[34] Daniel Snashall and Dave Cliff. Adaptive-Aggressive Traders Don't Dominate. *Lecture Notes in Computer Science*, pages 246–269, 2019.

[35] Steve Stotter, John Cartlidge, and Dave Cliff. Exploring assignment-adaptive (asad) trading agents in financial market experiments. In *ICAART (1)*, pages 77–88, 2013.

[36] Jóna Elísabet Sturludottir, Sigríur Sigurardóttir, Marta Serwatko, Erna S. Arnardóttir, Harald Hrubos-Strøm, Michael Valur Clausen, Sigurveig Sigurardóttir, María Óskarsdóttir, and Anna Sigridur Islind. Deep learning for sleep analysis on children with sleep-disordered breathing: Automatic detection of mouth breathing events. *Frontiers in Sleep*, 2, feb 17 2023.

[37] Tianyuan Sun, Dechun Huang, and Jie Yu. Market Making Strategy Optimization via Deep Reinforcement Learning. *IEEE Access*, 10:9085–9093, 2022.

[38] Gerald Tesauro and Jonathan L Bredin. Strategic sequential bidding in auctions using dynamic programming. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 2*, pages 591–598, 2002.

[39] Gerald Tesauro and Rajarshi Das. High-performance bidding agents for the continuous double auction. In *Proceedings of the 3rd ACM Conference on Electronic Commerce*, pages 206–209, 2001.

[40] Perukrishnen Vytelingum. *The structure and behaviour of the continuous double auction*. PhD thesis, University of Southampton, 2006.

[41] Bernard Widrow and Marcian E Hoff. Adaptive switching circuits. Technical report, Stanford Univ Ca Stanford Electronics Labs, 1960.

[42] Aaron Wray, Matthew Meades, and Dave Cliff. Automated Creation of a High-Performing Algorithmic Trader via Deep Learning on Level-2 Limit Order Book Data. *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, dec 1 2020.

[43] Bee Wah Yap and Chiaw Hock Sim. Comparisons of various types of normality tests. *Journal of Statistical Computation and Simulation*, 81(12):2141–2155, 2011.

[44] Zihao Zhang, Stefan Zohren, and Stephen Roberts. Deeplob: Deep Convolutional Neural Networks for Limit Order Books. *IEEE Transactions on Signal Processing*, 67(11):3001–3012, jun 1 2019.