School of Computing
FACULTY OF ENGINEERING AND
PHYSICAL SCIENCES

UNIVERSITY OF LEEDS

Final Report

TradeAI: Advancing Algorithmic Trading Systems with Deep Learning for Cryptocurrency Data

Tomás Zilhão Borges, Andrzej Miskow, Diogo Nuno Fernandes Gonçalves, Alexander Pitsin, Oleh Chernilevskyi, James Bridge

Submitted in accordance with the requirements for the degree of MEng Computer Science

2022-2023

COMP5200M Group Project

The candidate confirms that the following have been submitted.

Items	Format	Recipient(s) and Date
Final Report	PDF file	Uploaded to Minerva (12/05/23)
Link to Code Repository	URL	Sent to supervisor and assessor (12/05/23)
Link to Video Demo	URL	Sent to supervisor and assessor (12/05/23)

The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

(Signature of Students) Tomás Zilhão Borges, Andrzej Miskow, Diogo Nuno Fernandes Gonçalves, Alexander Pitsin, Oleh Chernilevskyi, James Bridge

Abstract

Integrating deep learning methods into algorithmic trading systems is advancing the financial industry, enabling sophisticated analysis and decision-making capabilities previously limited to institutional investors, making them accessible to retail investors. However, the application of deep learning in medium-frequency trading (1-30s), particularly in the cryptocurrency market, remains largely unexplored. This research aims to bridge this gap by investigating the feasibility and effectiveness of integrating deep learning methods into an algorithmic trading system specifically tailored for cryptocurrency trading. This study presents a comprehensive full-stack algorithmic trading system that integrates deep learning methods for trading strategies. Specifically, we delve into utilising novel transformer architectures, including Informer, Pyraformer, and an enhanced original transformer, into predicting cryptocurrency prices. Our research findings showcase these transformer models' superior performance compared to traditional ARIMA models, particularly when operating on larger datasets. Notably, the Pyraformer model exhibits good predictive accuracy while maintaining efficient training and inference times. Moreover, we seamlessly integrate these predictive signals into the environment definition of a Deep Reinforcement Learning (DRL) model, enabling effective order generation and decision-making. The findings contribute to understanding transformer models' effectiveness in medium-frequency cryptocurrency price prediction and provide a promising architecture for future research and development of trading strategies in this evolving field.

Acknowledgements

The team would like to express sincere gratitude to Dr. Nabi Omidvar for providing invaluable guidance and support throughout the lifetime of this project. Dr. Nabi's excellent mentorship, thorough feedback and genuine care for the team members has been instrumental for the success of the project.

Contents

Chapter 1

Introduction

The traditional image of a large room with continuously updating monitors on the walls and a floor filled with hectic crowds of traders repeatedly shouting "buy" and "sell" is gradually being replaced with one of quietly humming computers. In addition, complex algorithmic trading strategies can now execute multiple transactions at the speed of fractions of a second and are already responsible for around two-thirds of all stock market trading volume [99].

The feasibility of successfully creating an algorithmic system using Deep Learning models in its structure is first addressed. The market is a place filled with unpredictability and multi-variability, and accommodating an accurate and generalizable representation into a Deep Learning model is challenging. In addition, Deep Learning methods have only recently begun to be deployed to production and very rarely are used in latency-critical contexts such as algorithmic trading. Secondly, retail investors play a more significant role these days than ever before (eg. the GameStop saga), so one “bulls” the question of whether it is possible to make Deep-Learning-based algorithmic trading available for retail usage and user-friendly.

This paper introduces a full-stack algorithmic trading system for implementation of machine learning to the realm of medium-frequency (1-30s) cryptocurrency price prediction and order generation. Firstly, we demonstrate how limit order book cryptocurrency can be extracted via an external API and its subsequent processing for utilisation by deep learning models.

Specifically, this study showcases the application of Transformers for direct price predictions in cryptocurrency. We deploy the original Transformer model[105] and introduce two innovative transformers, namely Informer[113] and Pyraformer[92], specifically designed for time-series forecasting tasks. Additionally, leveraging recent advancements in positional encoding techniques[113], we enhance the original transformer model and fine-tune the parameters of each model to cater to the unique requirements of cryptocurrency price prediction. Furthermore, we investigate the impact of different dataset sizes to ascertain the relationship between dataset size and predictive performance. Through these experiments, we aim to advance the understanding and capabilities of Transformers in the domain of cryptocurrency price prediction.

Moreover, our approach leverages the output from predictive transformer models, denoted as alpha, through a Deep Reinforcement Learning model and a heuristics-based strategy for trading decision-making. This unique exploratory approach represents a significant contribution to the field, as it has not been previously explored in the literature for stock market or cryptocurrency data. Our results critically analyze the potential of this approach in medium-frequency trading (1-30s), positioning our work as a valuable contribution to the field and opening avenues for further research and development.

Furthermore, we integrate the above-stated methods into a novel algorithmic trading system (main deliverable) - TradeAI - that adopts an MLOps [40] development cycle to ensure the reproducibility and reusability of our work with new data and models. In addition, it is capable of backtesting different strategies and working as a server. Finally, the system is containerized and deployed to Azure, and a client interface is developed to interact with it in an algorithmic-trading-as-a-service fashion, allowing real-time monitoring.

1.1 Problem Statement

1.1.1 Aims and Scope

- To explore the feasibility of developing a full stack Algorithmic Trading System capable of running Deep-Learning-based trading strategies on Cryptocurrency limit order book data, keep track of the results and communicate them to a client through an interactive User Interface

1.1.2 Objectives

- Develop a comprehensive literature review that confirms the viability of the proposed algorithmic trade systems while outlining novel approaches in deep learning for price prediction and trade execution.
- Outline how order book cryptocurrency data can be acquired and processed for deep learning predictive models
- Implement various deep learning models for price prediction
- Explore the feasibility of Deep Reinforcement Learning strategies for order generation
- Develop an algorithmic trading system capable of executing multiple trading strategies using a combination of the implemented methods.
- Create a web client that facilitates communication with the algorithmic trading system and monitors the system's performance.
- Evaluate the performance and feasibility of price prediction using the implemented models and assess the effectiveness of financial trading within the developed system.

1.1.3 Deliverables

The project will deliver 1) a Git repository containing the project code, 2) a team report, 3) an individual report for each team member, 4) individual logbooks, and 5) a demo video showcasing the implemented system.

1.1.4 Ethical, Legal and other issues

For an in-depth look at the identified ethical issues of the creation of an application like this one, please refer to Appendix A.

1.1.5 Project Management

For an in-depth look at how Project Management is addressed, please refer to Appendix B.

Chapter 2

Literature Review

2.1 Algorithmic Trading Systems

Successfully developing trading algorithms and systems capable of consistently beating the market has been a regular topic of research since the rise of electronic trading in the 1990's and it has followed the trend of adding levels of abstraction between human decision making and actual trade execution. It is difficult to come across literature that precisely constructs systems employing deep learning-based price prediction as the alpha, which is subsequently integrated into additional deep learning techniques for order generation. However, it has been observed that over time, these systems have maintained a set of components that continually undergo reinterpretation in order to incorporate new technologies . Here, these "black boxes" [79] are dissected into their individual components, allowing for an understanding of how they function and how to evaluate the performance of a system.

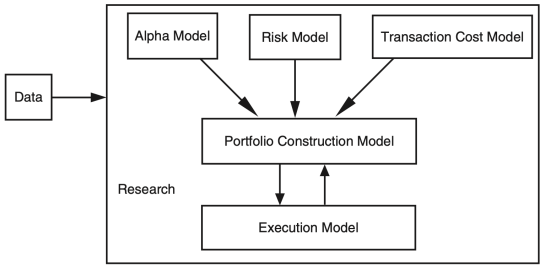


Figure 2.1: Algorithmic Trading System Architecture [79]

Data Handler is responsible for managing and processing market data [53] [54] [79]. Such data can be of real-time or historical nature, where the latter can be simulated into the system for backtesting (check Section 2.1.6). The data handler processes this information and disseminates it to the other components of the system.

Alpha Models are the "optimistic" part of the system and are usually considered the core of a trading strategy, as they're responsible for finding an 'edge' in the market, ie, systematically adding skill in timing and sizing the investment such that it is profitable [79]. Furthermore, "alpha is the portion of the investor's return not due to the market benchmark (beta)" [79].

Types of Alpha Models: Alpha models can be broadly classified into theory-driven and data-driven categories. Theory-driven models generalize market observations into hypotheses, while data-driven models gather observations to obtain predictive behavior, regardless of rational explanations [79]. Rishi K Narang (2009) [79] further divides theory-driven models into price-related and fundamental/sentiment data strategies. However, categorizing alpha strategies can be challenging for complex or data-driven solutions.

Theory-based fundamental strategies include E/P ratio-based (yield) investing, which focuses on buying undervalued stocks, considering markets tend to overestimate risk in risky instruments [79]. Growth strategies rely on historical growth levels for a security, while sentiment strategies depend on public opinion/news, both without guaranteeing valuation or yield [79].

Regarding price-related strategies, trend-following is based on the idea that price trends persist in the short to medium term [30]. Mean reversion assumes that asset prices fluctuate around a "center of gravity" in the short term due to supply and demand imbalances [79]. Pairs trading, a data-driven approach,

exploits market inefficiencies and mean-reverting behavior by trading pairs of historically correlated securities [45] [42]. Arbitrage seeks to find and exploit market patterns, often involving multiple securities and high-frequency trading (HFT) [73]. Market Making, though not a traditional data-driven alpha strategy, captures the bid-ask spread and profits from liquidity provision. It is often used in HFT [76].

Variables in Alpha Model Implementation: When implementing alpha strategies, several variables contribute to their distinctiveness. Firstly, alpha models can predict the direction, magnitude, and/or duration of a movement, along with an estimation of confidence or probability [79]. Secondly, the prediction horizon is essential, with strategies classified into Low-Frequency Trading (LFT), Medium Frequency Trading (MFT), and High-Frequency Trading (HFT). Balancing run frequency with prediction horizon is crucial due to per-trade transaction costs [79].

The alpha model definition is the most significant source of implementation variability and depends on the factors mentioned above. For instance, moving-averages models compute the average price over a specified period to smooth price fluctuations and identify trends [16]. Breakout strategies involve trading on price movements that break through a previously established range or trend [72]. Another variable is instrument selection. Alpha models can be relative to an instrument itself or an instrument relative to others. Liquid instruments are generally preferred as they offer more high-quality data and are more conducive to forecasting [79].

Machine learning (ML) has vastly expanded the variety of data-driven methods for seeking alpha and has extended to all parts of algorithmic trading systems. Zineb Lanbouri and Said Achchab (2020) [114] use LSTMs for stock price prediction at various time horizons. However, complex ML models in HFT are less common due to their high inference time. To overcome this, Andrew Boros et al (2017) [21] implement an HFT system on an FPGA, allowing professionals to use C++ strategies, including machine learning. Others leverage distributed and parallel computing for facilitating ML in low-latency systems.

Section 2.3 details LSTMs and Transformers for price prediction (explained as alpha in Chapter 3) and Section 2.4 covers Deep Reinforcement Learning for order generation.

Transaction Costs: In algorithmic trading, particularly high-frequency trading (HFT), transaction costs are crucial due to the high volume of trades. Profits may be eroded even if alpha is found. Obvious costs include commissions and fees, while hidden costs involve slippage and market impact. Slippage, resulting from latency, occurs when the price changes between the decision to trade and order execution [79]. Market impact, which is difficult to predict, arises when an order submission influences the instrument's price and is often related to the order size and liquidity available at the time [79].

Risk Model and Portfolio Construction: The risk model counterbalances the alpha model's market exposure, improving return consistency by determining trade sizes for each signal [47] [79]. Volatility (uncertainty) and dispersion (portfolio diversification) arise from standard deviations of instruments over time and between instruments, respectively, and can be proxies for risk [79].

Simple risk management involves a fixed amount per trade [47]. Ideally, leverage is adjusted for constant risk-adjusted returns. Traditional methods include value-at-risk (VaR) and expected shortfall (ES). Modern portfolio theory (MPT) framed resource allocation as an optimization problem, while the Black-Litterman model incorporated market factors into MPT [50]. Recently, machine learning (ML) methods have been applied to assess risk in market and portfolio management [110].

Execution Model: The execution module manages order submission and trade lifecycles. Market orders offer immediate execution at uncertain prices, while limit orders ensure the execution price but may not be fully or partially executed [47]. Execution algorithms strive to minimize market impact and transaction costs [18]. Recent research on high-frequency trading (HFT) has emphasized optimizing execution algorithms to reduce latency for capturing short-lived arbitrage opportunities [76].

Backtesting and Methods of Evaluation: Backtesting simulates market conditions to evaluate trading algorithms, using frameworks like event-driven and vectorized approaches [53] [54] [22]. High-frequency trading (HFT) backtesting is particularly complex due to data granularity, execution speed,

and HFT strategy complexity. Techniques addressing these include market microstructure models and liquidity constraints [89] [18] [82].

However, backtesting may not reliably predict future performance due to potential overfitting (when the model is excessively tailored to historical data), data drift (when changing market conditions cause strategy's performance to decline), lookahead bias, or the backtester's inability to mimic real-world situations accurately.

Performance evaluation of algorithmic trading involves various indicators and risk-adjusted return measures. Crucial metrics include absolute and relative returns, the Sharpe and Calmar ratios, maximum and average drawdown. For HFT, order latency is key, encompassing feed latency, order entry latency, and order response latency [82].

Market Considerations When Using Algorithmic Trading Systems: Algorithmic trading's market impact has sparked controversy and mixed opinions. It's been linked to market disruptions, like the 2010 Flash Crash, leading to worries about market manipulation, systemic risk [36], and increased regulation [86]. Its effect on liquidity and market efficiency is also debated. Some argue that high-frequency trading (HFT) boosts liquidity by reducing bid-ask spreads and price impact [100] [59], while others suggest HFT-provided liquidity can vanish in market stress periods [20].

Today's HFT space is dominated by entities with superior trading latency, as execution delays can negatively affect HFT. Hedge funds with proprietary software typically compete, leaving little room for retail algorithmic trading. As an alternative, Medium Frequency Trading (MFT) allows the use of similar strategies but with lower for technology investments [15].

Contrary to common belief, Hilbert and Darmon (2020) [56] indicate that more predictable structural patterns can increase uncertainty. Thus, a trading strategy's goal of local market predictability could lead to macro-level uncertainty and complexity. Therefore, it's crucial to recognize the potential instability and unreliability of algorithmic trading systems, which can generate skewed predictions, non-optimal trade orders, overfit historical data, or suffer from data drift without regular retraining.

Nevertheless, given the precedent of successful forecasts with various methods, cautious optimism for developing alpha in this project is warranted.

2.2 Data

Cryptocurrencies are relatively new assets that have gained formidable traction among investors and traders. Popular examples of crypto assets are Bitcoin(BTC) - introduced in 2009 as one of the first financial protocols that adopts blockchain technology and Ethereum (ETH) - a peer-to-peer transaction-based state machine that not only serves as a cryptocurrency but also allows users to deploy arbitrary code thus enabling a whole ecosystem of decentralized financial and non-financial applications to emerge. Adopting decentralized technology has been met with both speculation and optimism, therefore, creating a market for cryptocurrencies that has shown to currently be more volatile than traditional asset markets [43], Figure D.1. The challenges of gathering data to analyze the cryptocurrency market are discussed in [17] where the authors of the paper conclude that extracting cryptocurrency data, with the goal of portfolio optimisation and developing trading strategies and applications, should be done through exchange providers that directly reflect the state of the market, such as Coinbase [6] and Binance [1] rather than data aggregation platforms such as CoinMarketCap [7].

2.2.1 Limit orderbook data

In the context of developing applications that adopt machine learning algorithms to predict cryptocurrency market price - the authors of research paper [19] use LSTM neural networks by gathering data over a period of 180 day where data is extracted from CoinMarketCap and a multivariate model is developed

using the Open, Close, High, Low and Volume markers. The aim of this project is to shift towards using machine learning to develop higher frequency trading strategies that aim to accumulate profit in the short term. For that purpose, extracting limit order book data would offer more granular insight into how a particular market is moving compared to aggregated data over a time period (minutes, hours, days). Moreover, in the case of a live-trading application that makes predictions and executes trades, live information of the state of the market is available through the engine exchange platform [3] and commonly that is a limit order book engine. Limit order book is the term used to describe an order matching mechanism that accepts, cancels and updates ask and bid orders, where every ask or bid order is submitted together with a price and quantity of the asset that's to be bought or sold. Table 2.1 displays how an order book would look like at a particular moment of time.

Table 2.1: Limit order book format

timestamp	Side	Price	Qty
1671840447491	ask	16781.50	8149
1671840447491	ask	16781.90	2858
.	.	.	.
1671840447491	bid	16780.30	2443
1671840447491	bid	16781.40	28524

A study on high-frequency limit order book data modeling and prediction with the use of Support Vector Machines [63] identifies the median price, also referred to as midpoint, between the lowest ask and highest bid and the bid-ask spread crossing as metrics that can be used independently or combined to derive a trading strategy that could result in profit, moreover, the authors present thorough feature sets based on the respective price and volume of asks and bids. The authors of [83] build upon the work of [63] and further compel existing literature on high frequency trading research methods that predict median price. They define and compute a limit order book baseline dataset for five stock assets, following data representation suggested by [63] and compute cover over 10 days of data. Furthermore, the authors conclude from [31] that computing ask/bid levels yields moderate extra knowledge about the price of an asset. Drawing upon this research, the current project aims to extract features such as midpoint price between lowest asks and highest bids on a single level to avoid long feature vectors but also features that incorporate multiple ask/bid levels such as ask and bid median volume across several levels.

2.3 Methods for Time-Series Forecasting of Cryptocurrency Prices

Cryptocurrency price prediction, an area of keen interest, employs various time-series forecasting methods. Despite being an emerging field, valuable insights can be drawn from related studies on stock market predictions and time-series forecasting.

This section reviews literature and methods for predicting cryptocurrency prices using time-series analysis. We first discuss popular non-deep learning techniques like linear regression, ARIMA, and exponential smoothing. Then, we explore advanced deep learning methods like LSTM networks and transformers. Evaluating these techniques' strengths and weaknesses, we aim to offer insights for creating effective time-series models for cryptocurrency price prediction.

2.3.1 Non-Deep Learning Methods

Early approaches to financial stock market prediction mainly relied on classical mathematical and statistical models such as Hidden Markov Models or linear regressions, as well as some basic machine learning models like the Back Propagation Neural Network and Support Vector Machine (SVM) [111]. Similarly,

these techniques have been explored and analyzed in the cryptocurrency price prediction domain, as shown in an extensive survey [65].

Despite deep learning methods outperforming traditional techniques in univariate and multivariate settings, [49] [65], regression techniques are still widely utilised as a technical analysis metric to extract meaningful features from data. Recent research suggests that price-to-moving average ratios used in regression methods can be useful predictors of returns, particularly for assets with hard-to-value fundamentals, such as Bitcoin [38]. Autoregressive integrated moving average (ARIMA) and exponential smoothing are among the most commonly employed techniques for this purpose. These techniques offer some advantages over deep learning models, including simpler implementation and faster execution due to their lack of complex training processes. These advantages can be particularly beneficial when working with high-frequency data or real-time applications where lower latency is crucial[57].

Despite their usefulness, ARIMA and exponential smoothing are limited in their ability to handle sudden changes in the data and are unsuitable for long-term forecasting. To overcome these limitations, deep learning algorithms have been increasingly adopted in time series forecasting. Such algorithms can detect complex structures and relationships within the data and predict their effect [65]. Therefore, deep learning methods offer significantly improved results compared to traditional forecasting methods.

2.3.2 Deep Learning Methods

As previously noted, traditional time series forecasting methods have inherent limitations in modeling complex relationships within the data. Deep Learning algorithms have emerged as a promising alternative to overcome these challenges. This section aims to provide a chronological overview of various Deep Learning algorithms developed for time series forecasting, starting from early approaches and progressing to more recent and novel techniques that were explored in the development of our own application.

Recurrent Neural Networks(RNNs) are a type of neural network that is specifically designed to handle sequential data, making them a natural fit for time series analysis. While RNNs have been successful in capturing temporal dependencies, they suffer from the problem of vanishing gradients, which limits their ability to model long-term dependencies in the data. They've been used in multiple applications in the industry, although they are now mostly considered outdated for the intended use of this paper. The main issues of these networks are highlighted in [67], where it is explained that RNNs suffer either from vanishing or exploding gradients. To overcome this challenge, more advanced models such as Long-Short Term Memory (LSTM) and Transformers have been developed to better capture complex temporal patterns and dependencies in time series data.

Long Short-Term Memory (LSTM) has emerged as a replacement for standard Recurrent Neural Networks (RNNs) in sequential learning. LSTM addresses the vanishing gradient descent problem by utilising constant error carousels (CEC), memory cells, and gates. The memory cells store information and facilitate perpetual error flow, while gate units control data flow in and out of the cells[97]. LSTM has found success in finance, particularly in stock value forecasting [55], [27], [41]. Some LSTM models have shown effectiveness in predicting specific stock prices, while universal LSTM models have achieved 65-70% accuracy in medium-frequency stock trading [104]. However, experts have raised concerns about LSTM's limitations in handling complex financial data and capturing long-term trends [69]. LSTM's forecasting reach may be restricted by overspecialization to short-term fluctuations and a focus on small changes in absolute values rather than true predictive ability. To address these issues, stacking LSTM network [91]s and introducing a recurrent projection layer have been proposed to improve long-term accuracy and convergence. Nevertheless, LSTM models still face challenges in handling long-term dependencies. Attention-based LSTM models have emerged as a potential solution to overcome these limitations [115]. Further details on attention-based models are described in appendix C.1.

Transformers are a novel architecture in machine learning based around the encoder-decoder ar-

chitecture. They have emerged as a powerful architecture for processing long sequences of data. By incorporating a combination of multi-head attention and positional encoding, transformers can effectively capture complex temporal relationships and patterns, outperforming traditional models such as RNNs and LSTMs [105]. Furthermore, the modular architecture of transformers allows for highly parallelizable operations, enabling the processing of data in parallel and considerably reducing training and inference times. This unique characteristic of transformers has made them an attractive option for handling time series data, especially in scenarios where the efficiency of the model is critical [68].

Although transformers have demonstrated efficacy for various applications, their use for cryptocurrency data prediction remains largely unexplored. Existing research in this field primarily focuses on utilizing transformers for sentiment analysis on social media to aid in price prediction [46] [52]. While earlier studies on sentiment analysis have produced remarkable outcomes, the current study concentrates on utilising transformers for direct price prediction. To achieve this, we draw insights from research on applying transformers for stock market price prediction.

One research paper particularly relevant to our study is TransLOB, which predicts prices based on order book data using a convolutional network and a masked transformer module. Compared to other models, mainly LSTMS and RNNs, TransLOB performed significantly better on the same dataset and established a new benchmark for predicting price movements from limit order books. The authors have also highlighted that despite the $\mathcal{O}(N^2)$ complexity of the self-attention component found in transformers, their architecture is more efficient than existing LSTM architectures for this task [103] [102]. Furthermore, several research studies have applied transformers to stock market data, successfully predicting future closing prices on multiple exchanges worldwide [106] [78] [62]. These studies demonstrate that transformers outperform traditional deep-learning methods on price forecasting.

However, the self-attention operation used in transformers results in a quadratic memory and time complexity, which poses a challenge when processing long data sequences[108]. Furthermore, transformers face the challenge of potential temporal information loss due to the self-attention mechanism [112], which is a significant concern in time-series forecasting, where temporal relationships between data points are critical for accuracy. To overcome this, researchers have focused on developing time-series transformers that improve the modelling of temporal dependencies while preserving computational efficiency.

For instance, Informer [113] introduces a ProbSparse self-attention mechanism and a self-attention distilling operation to distinguish between significant and insignificant attention weights, resulting in an $\mathcal{O}(L \log(L))$ complexity. On the other hand, TransLog[51] uses a Logsparse mask to reduce computation complexity to $\mathcal{O}(L \log(L))$. At the same time, Pyraformer[92] utilizes pyramidal attention to capture multi-scale temporal dependencies hierarchically with an$\mathcal{O}(L)$ complexity.

Apart from improving the self-attention module, some time-series transformers use alternative decoder designs to address the original decoder's slow inference speed and error accumulation effects. For instance, Informer applies a generative-style decoder. At the same time, Pyraformer utilizes a fully-connected layer concatenating spatiotemporal axes as the decoder, which results in the lowest inference time among all time-series transformers [112].

Despite the improvements made to time-series transformers, there is still the potential for temporal information loss due to using self-attention variants in these models. This drawback is especially relevant for time-series forecasting.

2.4 Reinforcement Learning in Trading

Contrary to other algorithms in this report, Reinforcement Learning (RL) is not for forecasting but is action-oriented, benefiting from diverse predictive technologies.

Some of the earliest applications of Reinforcement Learning (RL) were built for financial markets. In [37], a detailed analysis of one such system is presented. This analysis was performed in 2004, and the

machine learning methods for policy creation were simplistic and outdated. Technology has progressed since then, and RL has also evolved.

Regardless of what method of RL is used, there is a set of core elements that define it as described, for example, in [87], where a financial system using Deep Reinforcement Learning was also built. This is a variation of this algorithm that will be discussed further in this paper. As a simplified explanation, there are 2 main components in a Reinforcement Learning algorithm: the agent, capable of making decisions and performing actions, and the environment, defined by the relevant information about the particular task we are adapting our agent to perform.

To enable an agent to interact with an environment, it requires a policy that dictates the actions to take in response to the observed environment. The policy can take any form based on the environment and actions that have been defined. Essentially, it is a set of rules or parameters that specify the appropriate action to take depending on the current state of the environment. The objective of the agent is to maximize returns based on the policy it follows. An ideal policy would generate ideal results, and it is often the task of the algorithm to approximate that policy. To achieve this, the impact of each action taken needs to be measured. This is most commonly done through a reward function – for example, the intent of the reward function can be to evaluate and define how profitable a purchase or sell might be in a financial system, in a chess game it could be to evaluate how likely the movement of a chess piece is to guarantee victory, etc. Every time an action is taken, the information about the environment is updated, and if training is being performed, the accuracy of the predicted reward is recorded. Based on this information, adjustments are made to the policy to optimize future actions. Figure F.1 taken from [101] illustrated in Appendix F showcases this interaction in a simpler way.

2.4.1 Deep Reinforcement Learning

In summary, "Deep Reinforcement Learning (DRL) is considered an approximate method that combines Neural Networks with RL" [60]. The most common approaches utilize Deep Q-Network (DQN) algorithms to derive and train the policy that the agent follows. Often, however, more complex approaches need to be considered depending on the problem. DRL algorithms can be separated into one of these categories depending on how the policies are generated, evaluated, and approximated:

Actor-Only approach, also known as Policy Gradient Methods, uses a neural network to derive approximate values for the policy. It estimates a gradient of the objective by maximizing rewards with respect to the policy parameters and adjusts them accordingly. This method is commonly used due to its simplicity and reduced training times [60].

Critic-Only approach consists of two main steps: Policy Evaluation and Policy Improvement. In the Policy Evaluation step, information is collected about the value functions of the policy to determine its effectiveness. In the Policy Improvement step, greedy actions are performed to improve the policy based on the results of the previous step. These two steps are repeated in an alternate manner until convergence is achieved. This approach tends to be more complex and challenging to implement [60].

Actor-Critic Approach, often referred to as a Hybrid Approach, combines the Actor and Critic approaches. The Actor selects actions at each time step to form a policy, and the Critic evaluates the actions taken by the Actor. The policy parameters are then gradually adjusted for the Actor to take actions that maximize the total reward based on the judgment provided by the Critic. This is the most complicated approach of the three and is also the slowest and most resource-intensive to train [60].

2.4.2 Usage of Deep Reinforcement Learning with Cryptocurrency

Cryptocurrency has not been around for a long time, and although there is abundant literature highlighting the usage of such algorithms in Financial Markets overall, for the Crypto market the research is limited. Still, in [90] a Deep Reinforcement Learning algorithm was developed for the Crypto market

in specific. And coincidentally, the data used as part of [90] is also quite similar to the data collected for the creation of our system. They opted to compare the performance of two different Actor-Critic approaches in different coins: the Advantage Actor-Critic (A2C) and the Proximal Policy Optimization (PPO). It also compared two different common reward functions: Positional PnL and Trade Completion. Finally, the algorithm was run and comparisons over the average daily returns were plotted on 3 different coins (depicted in table F.1 in Appendix F): Bitcoin (BTC), Ethereum (ETH) and LiteCoin (LTC). The comparative analysis provided by this paper provides great insight on future steps and approaches to be considered in the creation of our own DRL agent.

The nature of the Crypto market is very similar, however, to that of Stock and Forex markets. For these markets, there is extensive research on the advantages of this approach. In [101] for example, a Deep Reinforcement Learning agent using a simpler Deep Q-Network (DQN) was created and tested in the stock market. As a training set, they utilized data collected from the time frame of 2012 to 2017. For testing data, they used the time frame of 2018 to 2019. A benchmark trading assessment was performed between this strategy and other active and passive trading strategies, such as B&H, S&H, TF and MR. This assessment concluded with incredibly promising results, with the algorithm achieving the highest average performance among all strategies. Furthermore, between all 30 stocks listed, only two resulted in negative performance.

Likewise, in [60] a Deep Reinforcement Learning algorithm was developed for the same setting using a different strategy to DQN - a Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm. And as an addition, this algorithm was enhanced using Technical Indicators (such as the Relative Strength Index [RSI]) and Sentiment Data collected from headlines of news articles. A comparative analysis between the standalone algorithm and the algorithm using the indicators or the sentiment analysis data was performed, with incredibly promising results. The inclusion of technical indicators almost doubled the accumulated return, and the inclusion of sentiment data tripled it. Table F.2 taken from [60] within Appendix F showcases the actual figures benchmarked.

2.4.3 Sentiment Analysis for decision making

Cryptocurrency values are largely determined by public perception. For example, a tweet by Elon Musk caused a significant decrease in the value of Bitcoin [4]. Including sentiment analysis data can improve the returns of a Deep Reinforcement Learning algorithm [60]. StockGeist is a well-known platform for sentiment analysis data for cryptocurrencies, and they offer an API for easy usage [8].

Very little research on using Sentiment Analysis for Deep Reinforcement Learning algorithms in the context of cryptocurrencies, in particular, is available. However, further promising results have also been discussed in ([85]) where a general asset trading system called SentARL was developed using sentiment data. This approach also benefited from the A2C method discussed previously in this paper. In this example particularly, the addition of Sentiment Analysis also increased performance (although not to the same magnitude as in [60]). It can be concluded that sentiment data can be a great tool, but that its usefulness is highly dependent on the setting and the overall underlying design of the Deep Reinforcement Learning approach chosen.

2.5 System Deployment

Deploying machine learning (ML) projects has gained importance for organizations harnessing ML. A notable challenge is the gap between ML model development and real-world deployment [40]. This section delves into the foundations, principles and architecture of MLOPS, and how to deploy applications as microservices.

Factors that motivate MLOps and Main Components

MLOps foundations stem from challenges faced when taking ML systems to production. Complex ML systems can create technical debt, leading to increased maintenance costs, system fragility, and slowed innovation [35]. Traditional techniques for mitigating technical debt fall short for ML due to their complexities, including ad hoc ML bugs and extensive ML pipeline bugs, necessitating precise solutions for identifying underperformance and related bugs [96].

Effective ML asset management is essential for MLOps as well. Many professionals rely on traditional version control systems which fail to differentiate between key ML assets such as models and datasets, requiring specialized tools and practices diverging from traditional DevOps [40] [93].

Successful MLOps practices focus on speed, early validation, and maintaining multiple model versions to minimize production downtime [96] [94]. A mixed-method MLOps study identifies key principles which map into 9 technical components [40], addressing reported professional challenges. Components like "source code repository," "feature store system," "model registry," and "ML metadata stores" tackle asset management granularity. "Workflow orchestration component," "CI/CD component," "model training infrastructure," and "model serving component" standardize deployment. A "monitoring component" aids in identifying system-wide and model performance bugs.

MLOps Architecture

In practice, the MLOps lifecycle can be seen as a set of continuous sequential steps that can each achieve a high degree of automation [64] Google [48] presents how the technical components defined in the previous section may be organised in a system and propose 3 distinct maturity levels (0, 1 and 2) that take the system from fully Manual to fully automated. At level 0, every step is manual, the ML development pipeline is decoupled from operations, deployment is only focused on the prediction service with the final model and there's no live performance monitoring. Level 1 adds feature and model metadata stores (managing all data and all models) and ML pipelines for data and model validation (which means live re-training). Level 2 (Figure C.1) improves automation by implementing system-wide Continuous Integration and Deployment (CI/CD), allowing for rapid ML experimentation.

Similarly, J. B. Meenu Mary John & Holmström Olsson (2021) [74] propose a framework for companies to gradually start using MLOps. They propose automating 3 distinct pipelines over 4 stages. The Data Pipeline automates data collection and processing, the Modelling Pipeline automates re-trained model deployment and the Release Pipeline automates monitoring and creates triggers on performance degradation alerts. In addition, they perform 3 validation studies, one of them being a company that adopts MLOps by "packaging and deploying models using docker containers, using Kubernetes for automation and scaling, and tracking data, models, and experiments."

Another example, Li Erran Li et al [71] explain how their system works in UberEATS for the use case of Estimating Time of Delivery (ETD). A Machine Learning as a Service (MLaaS) platform is presented which is administrated by ML scientists via a User Interface (UI) and all use cases are exposed programmatically (via an API) to devices. It adopts a complete range of MLOps practices which would classify it as 'level 2" [48] and has a particular focus on good scalability. For instance, model training is organized in a tree structure (each node represents a distinct city) which optimizes fallback policies and edge cases. Re-training is done periodically and each service instance checks for new models and is responsible for its own packaging into a Docker container, validation and re-deployment. In addition, all model predictions are stored alongside actual outcomes.We explore the literature on deploying **Applications as Microservices in Appendix C.3**.

Chapter 3

Methods and Implementation

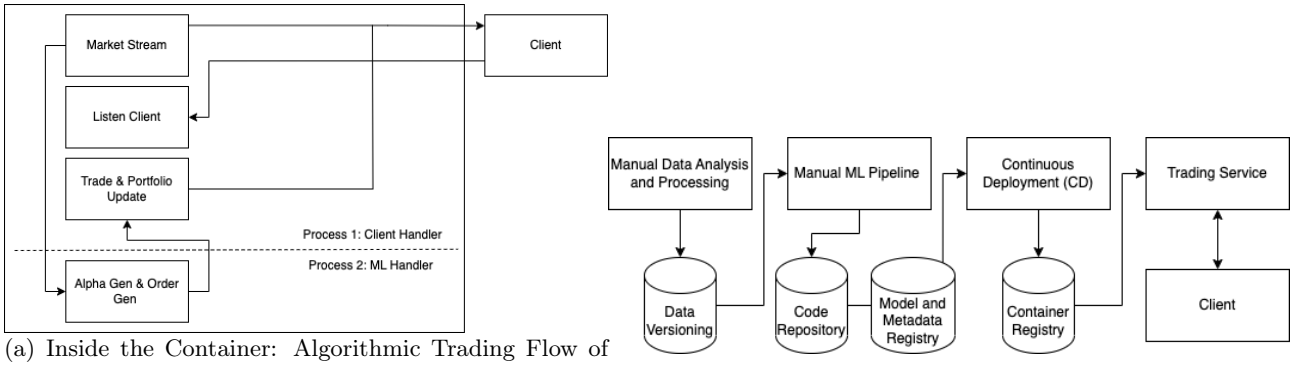
3.1 Project Architecture and Deployment

To provide a system where trading with ML methods can be executed, we 1) focus on allowing it to accommodate both the variability that comes from ML development and the stability needed for an algorithmic trading system, and 2) on effectively taking developed models into a production environment. To do so, we create our own MLOps pipeline (Figure 3.1b) based on the framework from Google [48] mentioned earlier. We can classify it mainly as level 0, as ML development is split from production; however, we can also identify features from level 1 by adding data and model registries.

In practice, data analysis and preprocessing (Section 3.2) are done manually and we use DVC for data version control and store all data artefacts in Azure File Share (AFS). This allows data to be readily available in the same place both for ML development and to be used in the microservice as a volume (permanent storage). Similarly, ML model development is done manually (Sections 3.3 and 3.4.1), MLFlow is used for keeping track of all trained models and their metadata, and Git LFS is used to store them inside Gitlab's remote server; the best-performing models are keept in the algorithmic trading service code repository (destined to be deployed), whilst the others and all metadata is stored in the wider scope of the full project repository. This, as well as keeping 2 sets of dependencies, is done to keep the algorithmic trading service as slim as possible.

Deployment is done by containerizing the algorithmic trading service (including ML models) with Docker Compose, which installs all of its dependencies, specifies the resources needed and mounts the AFS volume to the container. Furthermore, the container is then stored in Azure Container Registry (ACR). When a 'run' command is issued, an Azure Container Instance (ACI) is created to run it. This instance is publicly available and can be accessed from our developed client (Section 3.5) to execute and report our developed trading strategies. Each client needs an app instance with ideally 2 cpu's and 4 gb of RAM; these relatively high resource requirements are used to guarantee optimal machine learning inference time and trade execution with minimal overhead (see below).

Figure 3.1: Graphs for developed architecture



(a) Inside the Container: Algorithmic Trading Flow of Execution

(b) Project Architecture

3.2 Algorithmic Trading System

Inspired by the systems of Rishi K Narang (2013) [79] and Michael L. Halls-Moore (2015, 2016) [53] [54], an algorithmic trading system is developed (Figure 3.1a such that it uses the classic components reviewed in the literature and that additionally it has the capability of running performant machine learning inferences and working as a server. The system aims at accurately simulating the market with the use of a backtester and executing trading decisions on a medium-frequency range (MFT, 1-30s).

To do so, we divide the system into 2 processes and create a multiprocessing pipe as the communication channel between them. While the systems mentioned above as inspiration use synchronous event-based execution, the first process uses asynchronous communication between distinct routines, each with a particular role. After the connection with a client is established, the "Market Stream" coroutine is where everything starts by updating the client and the portfolio objects with market data on a second-by-second basis, and by periodically sending data to the second process to run ML predictions. The "Listen Client" coroutine reacts to any client-defined actions. And the "Report and Trade" coroutine executes any orders received from the order generation module and reports them to the client. We argue that even though the tasks of order execution and market updating are latency-critical, their execution is relatively quick and their presence in the asynchronous event loop shouldn't affect trading performance for the target case of MFT.

The second process is dedicated to handling machine learning tasks, which are usually CPU-blocking, so this separation into 2 processes is done to guarantee that they can be executed as fast as possible and that they don't block other latency-critical parts of the system. In addition, this second process is multithreaded to allow the run of multiple predictions concurrently and we initialize a pool of threads to reduce the overhead related to their creation and deletion. The multithreaded "Alpha Gen & Order Gen" routine receives market data from the first process. Then, it uses it to generate price predictions based on transformer models which are then used as alpha to make trading decisions either based on a deep reinforcement learning model or based on heuristics; finally, they're sent back to the first process as stated above.

Each developed component plays a key role and is used extensively in the aforementioned process. The data handler object is used to load the user-selected historical dataset, representing an orderbook converted to 1s data, and for providing a lookback context window the machine learning models can use to make educated predictions based on past data.

The created portfolio object manages time and a client's holdings. It records positions (BTC) and all USD holdings (BTC, cash, fees) every second. Additionally, it incorporates risk management by limiting maximum BTC holdings (default 1 BTC) and maximum BTC per trade (default 0.01 BTC for an initial capital of 100,000 USD).

In regards to the chosen execution model, we default all orders to market orders (MKT) and attempt at simulating transaction costs by 1) including a commission on all transactions based on the base Binance Exchange transaction fee of 0.1% per trade [28]. And 2) attempting at simulating slippage by adding an additional latency of 250 milliseconds at order entry/execution time.

Price prediction is used as a proxy for alpha generation. Thus, we use a collection of transformer-based models (Pyraformer, Informer and Transformer) to predict the price of BTC 30 to 180 seconds into the future and based on a contextual lookback window of 180 seconds (Section 3.3). We argue that even if their accuracy is not 100%, we can still use their magnitude, direction and distance relative to present for successfully using price predictions as alpha.

Finally, the order generation module takes into account the aforementioned concepts from alpha, risk, portfolio and transaction models to produce orders to be executed. We provide both a deep reinforcement learning model (DRL) (see Section 3.4.1) and a heuristics-based strategy (Section 3.4.2) to use these factors and generate orders.

The developed TradeAI server allows the developed client (Section 3.5) to select the dataset the back-test should run on, select the desired transformer for price prediction/alpha generation, select the desired strategy for order generation (DRL or heuristics based) and when to start/stop trading. Communication is done via a permanent websocket connection due to its ease of use and capacity for handling a high rate of messaging; custom JSON messages are used to exchange data between the 2 ends (Appendix I).

3.3 Data methods

3.3.1 Collection and Storage

Limit order book data of cryptocurrency assets is gathered through the Binance exchange market with the use of their API service that offers latest and historic data [1], additionally, limit order book data, that is already transformed, is also collected from Kaggle [11] for testing. The dataset collected from Binance represents 7 days of Bitcoin trading activity where, after feature extraction, the dataset will be split in different configurations for training and testing of the machine learning models. Naturally, the data received from the exchange platform API requires a significant amount of storage space. Under these circumstances a data versioning control software tool such as DVC [9] assists with keeping the data separate from the project's repository by keeping a hash value corresponding to a data file within the project while storing the actual data in the cloud, thus, allowing for the project's repository to be more compact.

3.3.2 Feature extraction

After a request is sent to the API, containing the symbol of a cryptocurrency asset and the time period, a response is received that gives access to limit order book data in the format that is displayed in table 2.1 in Section 2. There are multiple entries for every timestamp that capture all the buy and sell orders at that particular time. Since the limit order book represents what were once real time events the difference between consecutive timestamps is only several milliseconds. To address that, starting from the first time field, timestamp ranges are computed that bundle together consecutive timestamps where each timestamp range represents 1 second of information. After this operation, the newly computed time ranges would be traversed iteratively and at that stage feature extraction can be done on every iteration for the corresponding 1 second range. At every step the lowest ask price and highest bid price are considered the 1-st level of the order book information that resides in the particular time range, the second lowest ask and highest bid would be level 2, the third - level 3, etc. Leveraging the literature review and considering the complexity of storing long property vectors, the features presented in Table D.1 are extracted from the limit order book information that resides in each timestamp range, where P denotes Price and V denotes Volume. The specific configuration of the data extracted is 1 level (n=1) for midpoint price, spread, best ask volume and 10 level (n=10) for the mean ask price, mean bid price, mean ask volume and mean bid volume. The final data format is displayed in 3.1. Finally, the data is split in training and test sets with a percent weight ratio of 80/20.

Table 3.1: Extracted features from limit orderbook data

timestamp	midpoint	spread	volume	mpa	mpb	mav	mbv
1671483146570	16596.76	0.10	41.69	16596.89	16596.55	37.44	3.74
1671483147570	16596.75	0.10	41.75	16596.80	16596.51	33.36	7.63
.	.	.	.				

3.3.3 Evaluation

To evaluate the correctness of the developed data transformation software that converts the limit order book into features, a historic dataset is extracted from Kaggle [81] that contains the midpoint feature while, for the same period of time, data in limit order book format is extracted from Binance and transformed. The resulting graph of the midpoint values from the transformation is compared side to side to the Kaggle midpoint data and are presented in Figure D.2. The comparison confirms that the developed by the team feature extraction script computes accurately the trends within the data.

3.4 Predictive Models

Predictive models are a powerful tool for accurately predicting future events or outcomes based on historical data. With the increasing availability of large and complex datasets, advanced machine-learning techniques have become essential for extracting meaningful insights and making informed decisions. This study conducted an extensive literature review to identify a range of predictive models, including the Transformer, Informer, and Pyraformer. These models have demonstrated impressive performance in learning complex patterns from historical data and accurately forecasting future trends.

This section provides a comprehensive exploration of the implementation details and functionalities of transformer models, focusing specifically on their application in forecasting Bitcoin prices within the realm of cryptocurrency data. The appendix, labelled "Data Preparation" (Appendix E.1), delves into the necessary pre-processing steps to apply order book data to transformer models.

3.4.1 Transfomers

The aim of this section is to provide a comprehensive exploration of transformers, their core ideas, architecture, and recent variants in the context of time-series forecasting of Bitcoin currencies. The section begins by discussing the fundamental concepts of the original transformer model and how it can be applied to time-series forecasting. Subsequently, the latest transformer variants designed explicitly for time-series forecasting, are investigated to evaluate their potential for improving Bitcoin currency forecasting and the enhancements they bring over the original transformer model.

Transformers Core Ideas

Self-attention is a pivotal component of the Transformer architecture, facilitating the capture of relationships among all elements within an input sequence. It involves computing a weighted sum of the input values by evaluating the similarity between the query vector and the key vectors for each element. This is achieved through the dot product between the query vector and the key vectors of all sequence elements, resulting in a similarity vector of length N. Subsequently, a softmax function generates attention weights for each element based on the similarity vector. Finally, the value vectors of the elements are multiplied by their respective attention weights and summed, yielding the weighted sum. The self-attention mechanism can be represented mathematically as shown in Equation 3.1, where Q, K, and V denote the query, key, and value matrices, respectively, and d_k represents the dimension of the key vectors.

$$\text{Self-Attention } (Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{3.1}$$

Multi-head attention

is an extension of self-attention utilized in the vanilla Transformer architecture. The primary goal is to capture multiple representations of the data simultaneously, which is achieved by employing several

parallel self-attention layers. To begin, the input sequence is split into multiple heads, and each head independently and simultaneously processes the input using self-attention layers. Specifically, each head's query, key, and value matrices are transformed using learned linear projections denoted as W_i^Q, W_i^K, and W_i^V, respectively. Subsequently, each head applies the self-attention mechanism as described in equation 3.1 to compute its attention weights. Once all heads have computed their attention weights, the resulting weights are concatenated and passed through a linear layer W^O to generate the final output of the multi-head attention layer. This output is represented in the equation 3.2 where the output of the i-th head of the multi-head attention mechanism is denoted by head_i.

$$\begin{aligned} \text{MultiHead}(Q,K,V) &= \text{Concat}\left(\text{head}_1,\ldots,\text{head}_\text{h}\right)W^O \\ \text{where}\ \ \text{head}_\text{i} &= \text{Self-Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right) \end{aligned} \tag{3.2}$$

One benefit of multi-head attention is that it can more effectively model long-range dependencies in the input sequence. By splitting the input into several heads, each focusing on a different aspect of the input, multi-head attention can more easily capture long-range dependencies that would be difficult to capture with a single self-attention layer. Additionally, the parallelization of attention heads means that computations can be performed in parallel.

Positional Encoding is a technique used in the Transformer architecture to inject positional information into input sequences. Since transformers don't use recurrence and convolutions, they do not have an inherent notion of order. For transformers to use the order of the sequence during training, information about the relative or absolute position of the data in the sequence must be injected.

There are different approaches to implementing positional encoding for time series data. The original transformer uses sine and cosine functions to encode the position of each element in the sequence, as seen in equation 3.3 where *pos* refers to the position in the sequence, i is the index of the encoding dimension, and d_{model} is the dimensionality of the model. Although this approach provides local positional context, it does not fully leverage global information of time series data, such as the year, month, day, and time and specific events or holidays [113]. These global features are critical for capturing long-range dependencies in time series data.

$$\begin{aligned} PE_{(pos,2i)} &= \sin\left(pos/10000^{2i/d_\text{model}}\right) \\ PE_{(pos,2i+1)} &= \cos\left(pos/10000^{2i/d_\text{model}}\right) \end{aligned} \tag{3.3}$$

To solve this problem, Informer[113] implemented positional encoding that captures local and global dependencies in time series data. To capture local dependencies Informer model utilises the positional encoding implemented in the vanilla transformer but improves it. Informer changed the denominator in the positional encoding formula from a fixed value of 10,000 to a variable value computed as $(2L_x)^{\frac{2j}{d_\text{model}}}$, where L_x is the length of the input sequence and j ranges from 0 to $\frac{d_\text{model}}{2}$. This allows the positional encoding to be tailored to the input sequence's specific length, improving the model's performance by capturing local dependencies more effectively. The modified formula is shown in Equation 3.4.

$$\begin{aligned} \text{PE}_{(pos,2j)} &= \sin\left(pos/\left(2L_x\right)^{2j/d_\text{model}}\right) \\ \text{PE}_{(pos,2j+1)} &= \cos\left(pos/\left(2L_x\right)^{2j/d_\text{model}}\right) \end{aligned} \tag{3.4}$$

To effectively capture the global dependencies, Informer uses learnable temporal embedding, also called "stamp embedding." The temporal embedding is a layer representing each global timestamp in the input sequence with a learnable embedding vector of a finite set of possible values. Temporal embedding uses positional embedding found in equation 3.4 on each part of a timestamp, such as a year, month, day, hour, and minute to capture the temporal information of the input data. This allows the model to consider the order and time-dependent relationships between the input data and helps it better capture

long-term dependencies in the data. The original implementation only included embedding to as low as minutes; however, since we are dealing with second data, this has been modified to also include seconds.

An experiment examined a transformer-based model's most suitable positional encoding technique by testing a data sample using the original transformer architecture. Table 3.2 presents the results, indicating that the Informer positional encoding technique outperformed the original. Moreover, incorporating global encoding into the Informer technique further enhanced the model's performance.

Table 3.2: Results of different positional encoding techniques trained on vanilla transformer

PE Type	MSE	MAE	Avg Train time (s)
Vanilla PE	0.063	0.159	3.752
Informer PE	0.054	0.122	3.852
PE + Temporal	0.052	0.120	9.440

Since all time-series transformers utilize the positional encoding from the Informer model, the same positional encoding method was applied to the original transformer to ensure fairness and comparability in the evaluation.

Transformer Architecture

The Transformer architecture consists of two key components: the Encoder and the Decoder.

Encoder is responsible for generating hidden representations of an input sequence. These representations summarize the meaning of each token in the context of the entire sequence and are subsequently fed as input to the Decoder module. The Encoder module comprises a stack of identical layers, each containing two sub-layers: a multi-head self-attention mechanism and a fully connected feed-forward network. The multi-head self-attention mechanism enables the Encoder to capture global dependencies between the elements of an input sequence. At the same time, the feed-forward network introduces nonlinearity as the output from the multi-head attention mechanism is linear.

Decoder module of the Transformer architecture generates an output sequence based on the hidden representations generated by the Encoder module. Like the Encoder, the Decoder module comprises a stack of identical layers, each with two sub-layers from the Encoder module and an additional third sub-layer performing multi-head attention over the output of the Encoder stack. The self-attention sub-layer in the decoder stack is modified to prevent positions from attending to subsequent positions. The output embeddings are offset by one position to ensure that the predictions for position i depend only on the known outputs at positions less than i.

Informer

Informer[113] is a modified version of the Transformer architecture incorporating three key changes: the ProbSparse self-attention mechanism, the self-attention distilling technique, and the generative-style decoder. These modifications enable Informer to achieve a time and memory complexity of $\mathcal{O}(N \log N)$, substantially improving the original Transformer architecture. Furthermore, the number of decoder operations is reduced from L to $\mathcal{O}(1)$, where L represents the sequence length. Specifically, the ProbSparse self-attention mechanism and self-attention distilling technique allow for efficient computation of attention weights. At the same time, the generative-style decoder provides a more effective approach for generating output sequences.

ProbSparse self-attention mechanism is a modification to the self-attention mechanism used in the Transformer architecture, designed to reduce the attention mechanism's computational complexity and memory requirements. The authors of the Informer paper discovered that the self-attention output could be represented as a probability distribution based on the probability distribution $p(k_j|q_i)$ of attention

scores calculated in equation 3.1. Such distribution generates a long-tail distribution of attention scores, where only a small subset of dot-product pairs contributes to the attention mapping. In contrast, most pairs produce negligible attention weights that can be disregarded.

Therefore to distinguish the important queries, the authors employed Kullback-Leibler divergence to measure the likeness between the two probability distribution of attention scores p and a hypothetical uniform distribution of weights q. This is because in a case where the attention scores follow a uniform distribution, where all key vectors are assigned roughly the same weight, the output of the attention mechanism becomes a sum of the value vectors, which can be represented as $q(k_j|q_i) = 1/L_K$. In such cases, the attention mechanism does not effectively capture the relationship between the query and key vectors. It adds no value to the model. Using this measure, they identified the dominant dot-product pairs that encourage the corresponding query's attention probability distribution away from the uniform distribution. The equation to calculate the sparsity measure is shown in equation 3.5

$$\bar{M}\left(\mathbf{q}_i, \mathbf{K}\right) = \max_j \left\{ \frac{\mathbf{q}_i \mathbf{k}_j^\top}{\sqrt{d}} \right\} - \frac{1}{L_K} \sum_{j=1}^{L_K} \frac{\mathbf{q}_i \mathbf{k}_j^\top}{\sqrt{d}} \tag{3.5}$$

The approach randomly selects a subset of dot-product pairs ($U = L_K \ln L_Q$) from the long-tail distribution to calculate $M(q_i, K)$. Then, the Top-u pairs are chosen to create the sparse matrix Q, with the max operator applied to $M(q_i, K)$ to ensure numerical stability.

A larger value of $M(q_i, K)$ for the i-th query indicates that its attention probability distribution is more diverse and has a higher probability of containing the dominant dot-product pairs. Based on the proposed measurement, the authors proposed the ProbSparse self-attention mechanism, which allows each key to attend only to the top-u dominant queries presented in equation 3.6 where $\overline{\mathbf{Q}}$ is a sparse matrix that only contains the top-u queries controlled by a sampling factor c, where $u = c \cdot \ln L_Q$.Therefore ProbSparse self-attention only needs to calculate $\mathcal{O}(\ln L_Q)$ dot-product for each query-key vector achieving overall time and memory complexity of $\mathcal{O}(\ln L)$.

$$\mathcal{A}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Softmax}\left(\frac{\overline{\mathbf{Q}} \mathbf{K}^\top}{\sqrt{d}} \right) \mathbf{V} \tag{3.6}$$

Self-attention Distilling is a mechanism employed to address the issue of redundant feature combinations that arise as a natural consequence of the ProbSparse self-attention mechanism in the encoder's feature map. Distillation operation is used to keep important attention weights and remove insignificant ones. This operation involves forwarding the input from one layer of the Informer encoder to the next, with the goal of prioritising the superior feature combinations that have dominating features. The distilling operation includes a max-pooling layer that down-samples the output of each layer and progressively reduces the memory usage of the model. Using this technique, the self-attention distilling mechanism can efficiently handle extremely long input sequences and improve the model's performance in long sequence time-series forecasting. Table 3.3 summarizes the experimental results using the Informer model to evaluate different self-attention techniques.

Table 3.3: Results of Self-Attention techniques trained on Informer transformer

Type of Attention	MSE	MAE	Avg Train time (s)
Self-attention	0.054	0.123	**25.85**
Self-attention + Distil	0.075	0.174	31.06
Prob-Sparse	0.064	0.139	89.31
Prob-Sparse + Distil	**0.048**	**0.114**	38.19

Generative-style Decoder utilises a standard decoder structure that comprises two identical multi-

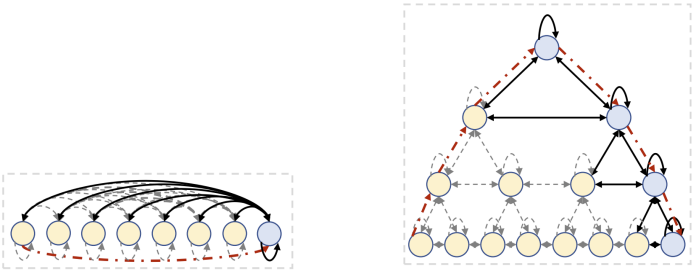
head attention layers. However, unlike the dynamic decoding process of the vanilla transformer, it uses generative inference to generate the output. In dynamic decoding, the decoder generates a sequence of outputs iteratively, one token at a time. The decoder receives the previous token and the encoder's output at each step. Then it generates the next token by computing the conditional probability distribution over the vocabulary of possible tokens given the previous context. This process continues until a special end-of-sequence token is generated resulting in L steps where L is the sequence length.

In contrast, the Generative-style Decoder adopts a different approach. Instead of selecting specific tokens, a sequence of length L_{token} is randomly sampled from the input sequence. For instance, for predicting the next 1 minute of Bitcoin prices, the last 300 seconds of data (5 minutes) could be selected as the start token. Subsequently, the decoder is fed with $X_{de} = \{X_{10min}, X_0\}$, where X_0 contains the timestamp of the target week, i.e., the next minute. This technique allows the decoder to generate the next 60 time steps using a single forward operation, largely outperforming the original decoder.

Pyraformer

Pyraformer[92] proposes a novel pyramidal attention-based Transformer that aims to capture long-range dependencies in time series data while maintaining low time and space complexity. The authors of Pyraformer state that calculating dependencies between different data points in a sequence can be modeled on a graph, where shorter paths between points lead to better dependency capture. An example of this can be seen in figure 3.3a, which demonstrates that the self-attention module has a max path of $\mathcal{O}(1)$ as each point attended to all the points at the cost of quadratic time complexity. In contrast, RNN and CNN models typically have a max path of $\mathcal{O}(L)$ but have a time complexity of $\mathcal{O}(L)$.

Figure 3.2: Graphs of model dependencies between data points [92]



(a) Full Attention (b) Pyraformer

Pyraformer proposes a transformer model with a time complexity of $\mathcal{O}(L)$ while achieving a maximum path length between $\mathcal{O}(1)$ data points, the same as the self-attention module employed in the Vanilla Transformer. As a result, the model can handle long input sequences and effectively model the dependencies between every data point. This is achieved via two modifications. The first modification replaces the original encoder with a Coarser-Scale Construction Module (CSCM) module, which formats the data into a hierarchical structure. The second modification replaces multi-head attention with Pyramidal Attention Module (PAM).

Pyramidal Attention Module (PAM) is a novel attention mechanism that extends the traditional self-attention mechanism of the Transformer model. The key idea behind PAM is to model the dependencies between different data points in a time series sequence using a pyramidal graph-based approach seen in figure 3.3c. This approach involves dividing the edges of the graph into two groups: inter-scale connections and intra-scale connections.

Inter-scale connections in the Pyraformer aim to capture global dependencies and patterns across longer time frames. These connections are structured as a C-ary tree, where each parent node has C children. Nodes at the lowest level correspond to the time points in the original time series, such

as second observations. In contrast, higher-level nodes represent more complex features like minute, hourly, and daily patterns. By building a multi-resolution representation of the original sequence through latent coarser-scale nodes, inter-scale connections facilitate the capture of long-range dependencies by connecting neighbour nodes via intra-scale connections. On the other hand, the intra-scale edges capture the local dependencies at each resolution by connecting neighbouring nodes. Thus PAM offers a concise representation for long-range temporal dependencies through the capture of patterns at coarser resolutions and the modelling of short local dependencies using sparse neighbouring intra-scale connections. This approach effectively reduces the computational burden associated with the original transformer model.

Coarser-Scale Construction Module (CSCM) is used to build the pyramidal graph structure seen in figure 3.3c. The CSCM is designed to take an input sequence and generate multiple coarser-scale versions of the original input. The CSCM comprises two main elements: a down-sampling layer and a feature extraction layer. The down-sampling layer reduces the input sequence's resolution by C, where C is a user-defined hyperparameter denoting the number of children per parent in the pyramidal graph. Subsequently, the feature extraction layer extracts features from each down-sampled sequence by performing convolutions on the children nodes. These are then used to create the coarser-scale nodes in the pyramidal graph.

The CSCM operates recursively to construct multiple levels of the pyramidal graph. Specifically, the output of the CSCM at level i serves as input to the CSCM at level $i + 1$, which generates an even coarser version of the input sequence. This iterative process continues until the desired number of levels is achieved. To decrease the number of parameters and computations, the module reduces the dimension of each node using a fully connected layer before inputting the sequence into the stacked convolution layers and restoring it after all convolutions. This bottleneck structure dramatically reduces the number of parameters in the module and safeguards against over-fitting.

3.4.2 Transformers Experiments

Optimal Hyperparameters Numerous experiments were conducted to determine the optimal hyperparameters for each transformer model. Each experiment comprised 20 epochs, and the early stopping criterion was set at a patience level of 3. To execute many experiments within a limited time, all trials were conducted on a preprocessed 3-hour order book dataset of Bitcoin prices. Furthermore, the experiments were conducted using single-variate data, a look-back window of 90 seconds, a label length of 60 seconds, and a prediction length of 30 seconds. Grid search was employed to explore the parameter space, and for each trial, Mean Squared Error (MSE) and Mean Absolute Error (MAE) scores were recorded. The Adam optimizer and MSE loss criteria, commonly utilized in time-series transformer papers, were utilized for each run to compare our findings with those in the literature. Lastly, MLFlow was employed to store the training and evaluation metrics for all the experiments. The table 3.4 contains the optimal parameters for each transformer model, full results can be found in the Appendix Table E.1.

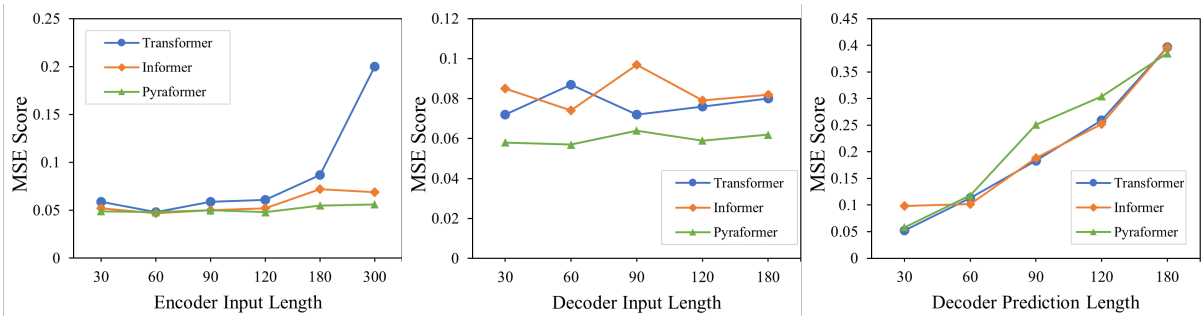
Table 3.4: Optimal Hyperparameters of Transformer, Informer and Pyraformer on Bitcoin Data

Hyperparameter	Search Space	Transformer	Informer	Pyraformer
Dimension of the model	{256,512,1024,2048}	1024	2048	1024
Number of attention heads	{2,4,8,16,32}	16	2	8
Number of encoder layers	{2,4,6,8}	2	2	-
Number of decoder layers	{2,4,6,8}	6	8	2
Dimension of feed-forward network	{512,1024,2048,4096}	1024	2048	1024

Parameter Sensitivity The present study investigates the prediction capabilities of transformer models with respect to the encoder input length, decoder input length, and output length. To this end, three experiments were conducted, each focusing on a different aspect of the transformer architecture.

In the first experiment, the study investigated the impact of varying the encoder sequence length on

Figure 3.3: The parameter sensitivity of three components in Transformer Models.



(a) Sequence Input length (b) Decoder Input Length (c) Prediction Length

the prediction performance of transformer models. The results of this experiment are presented in Figure 3.3a, which showed that vanilla transformers were unable to process long data sequences effectively. In contrast, time-series transformers, Informer, and Pyraformer demonstrated superior performance, maintaining good mean squared error (MSE) scores beyond 180 points. In the second experiment, the study examined the effect of varying the decoder input length on the prediction performance of transformer models. The findings are presented in Figure 3.3b, which demonstrated that the input length to the decoder was not a significant factor, particularly for the Pyraformer. Lastly, the study investigated the effect of a reduced input sequence of 90 points on prediction accuracy and its relationship with increased output sequence length. The results of this investigation are presented in Figure 3.3c, which indicated that none of the transformer models performed satisfactorily when the input sequence was short, implying that all models could capture the data sequence's dependencies. The time-series transformers exhibited lower performance than the vanilla transformers due to their intended design to process long input sequences. Therefore, it is crucial to ensure that the input sequence is long enough for time-series transformers to detect inter-point relationships and provide accurate predictions.

3.5 Trading Strategies

Two main approaches were explored as part of this report for the decision-making component. This section is going to provide a high-level description of the architectural choices and methodology of these strategies, as well as an explanation of how these strategies rely on the predictive component.

3.5.1 RL-Based Strategies

Although complex, it was prevalent in the literature that several Reinforcement Learning strategies could be beneficial for our application. Still, few research papers explored the usage of this technology within the Cryptocurrency market, and none were found that utilised data from predictive models as part of their environment definition. The following section of this report describes how RL was incorporated into this project and what decisions in architecture were made. Many of the approaches discussed are either quite novel or experimental.

Architectural Decisions

As it is the current industry standard for RL strategies across the industry, Deep Reinforcement Learning was selected. However, multiple variations of this approach exist of varying complexity and results.

Initially, DQL was considered. There were a few aforementioned reports that used this technology, and it is one of the simplest to implement whilst also typically scoring promising results. However, the

A2C approach showed incredibly promising results, outperforming DQL in the scope of this application (highlighted in table F.1). Ambition won the debate, and a choice was made - the A2C approach.

To develop an A2C algorithm, three distinct neural networks are required to be developed and calibrated to function collaboratively. The actor network produces the actions to be executed, while the critic network gauges the forthcoming rewards in the current setting. In addition, another critic network determines the advantage function, which is the contrast between the estimated critic rewards and real rewards, and learns to reduce the variance between them.

This incredibly complex puzzle is hard to balance, with multiple parameters that should be considered. Learning rates, optimizers (such as Adam, SGD, and RMSprop), policy update rules (such as Proximal Policy Optimisation), layer architectures, technical indicators, training batch sizes... all of these were experimented with. For the sake of brevity, the most interesting findings are described in Table F.4 in Appendix F. After much experimentation, it can be concluded that no convergence was observed. Higher learning rates quickly caused the model to overfit, evidenced by the actor's increased loss metric. Halving the model's layer sizes created worse results, and fundamentally changing the architecture (by, for example, removing the LSTM layers) did not cause significant differences in profits. It did, however, cause a lower yet more volatile figure for the actor loss calculation throughout training.

The chosen model was selected based solely on its lower monetary loss, as none of the models generated a profit. Its performance will be compared to heuristic-based approaches in the rest of this report.

To improve training times and increase convergence based on the results found, an Epsilon-Greedy approach was used as well as dropout layers throughout the design. The Epsilon-Greedy approach is a technique that selects a random action given a parameterized probability for the algorithm to take. It is often used to increase exploration of the network during training and avoid convergence in local minimums which results in sub-optimal solutions. This, however, did not fix the aforementioned problems nor did it significantly change any of the figures in Table F.4, Appendix F.

The final solution combines the usage of CNNs, LSTMs, and Fully Connected layers for each one of the networks. All 3 networks have a similar architecture, differing mostly on the last layers. The number of layers per network can be found in Appendix F, table F.3. For more details about their size and structure, please consult the linked git repository of this project.

Environment Definition

Defining the environment in a Reinforcement Learning agent is perhaps one of the most important tasks. The environment contains all the information the agent can have access to, so it is crucial to ensure its quality - poor or irrelevant information supplied can result in equally poor decision-making. It can be quite a complex task when simulated from scratch, so instead the Gymnasium library was used as the framework for its creation. Gymnasium is an open-source library created as a fork of Open AI's gym library, which is commonly used for the development of Reinforcement Learning approaches. The created environment for this project is built upon the most basic features of Gymnasium's "Env" class, with all the appropriate modifications to fit the intended reward function, state definition, and step function. As shown in [90], Trade Completion showcased the best results as a reward function. Policy Proximal Optimisation (PPO) was also selected as a policy update rule. For the final solution, our algorithm uses both strategies - having Trade Completion as the reward function and using a PPO strategy as the policy update rule instead of the traditional gradient descent approach. PPO is often chosen for promoting more stable policies and tends to perform well in environments with high volatility [12].

In [90], the usage of technical indicators instead of raw, normalized data was explored. Due to their results (discussed in the literature review section of this report), it was decided to take their software as an example and explore their usage. To that end, a custom RSI indicator function to represent the rate of increase/decrease of ask and bid prices was created. This metric was implemented to substitute

the usage of a look-back window of past data points to make a prediction. Equation 3.7 taken from [90] describes the mathematical logic behind this indicator.

$$\text{CRSI}_{\ t} = \frac{\text{gain}_{\ t} - |\text{loss}_t|}{\text{gain}_{\ t} + |\text{loss}_t|} \tag{3.7}$$

This algorithm not only predicts the past and future price indices but also incorporates information from predictive transformer models, a novel idea that has not been done in previous research. Unfortunately, due to time constraints, the significant complexity of the A2C approach, and the lack of sentiment data for the period of time for the data obtained, it was not possible to explore the usage of a sentiment indicator. Future research should be conducted to explore its usefulness.

Therefore, a state within this environment is defined as a tensor containing the following metrics: the three RSI indicators described above, the profit and reward of the last action taken, the current midpoint and spread value, and finally the current value of funds and stock that the agent currently holds.

3.5.2 Heuristic-Based Strategies

The Heuristics-based strategy is intended to explore ways of programmatically using and highlighting price predictions as alpha to make trading decisions and creating a baseline for the Deep Reinforcement Learning approach in the previous section.

The reload period, denoted as K, is the time interval (in seconds) between two consecutive prediction initiations. In this strategy, we set K equal to the transformer's chosen prediction length, P, to ensure that all trading decisions are made with no overlap and for P seconds away from the current time. This is achieved by utilising the most distant future prediction provided by the transformer.

Hence, every K seconds we calculate a possible net profit that will happen at the time that the prediction resolves by taking the difference between the current (t0) price and the price at the prediction time (t0+p). In addition, we take into account the commission involved for the portfolio's maximum quantity per trade, both as defined in Chapter 3. Finally, we take this possible net profit as a degree of confidence and use a threshold, T, such that if netProfit $>$ T: BUY, if netProfit $<$ -T: SELL, else HOLD.

$$\text{netProfit}_{t0+P} = \ \text{pricePrediction}_{\ t0+P} - currentPrice_{t0} - commission \tag{3.8}$$

3.6 Client Application

The front-end web application is a key component of the system and serves as the primary interface for user interaction. React, a popular framework is used to create a simple and user-friendly web app that seamlessly interacted with the back end. The overall design process involves user research, prototyping, and iteration, while the development process utilized various technologies and tools. The user interface design was focused on the visual and interactive elements, ensuring ease of navigation and functionality. Accessibility and a more detailed explanation of the application is explored in Appendix H.

.

Chapter 4

Results

This chapter presents the main results and discussion on both the topic of price prediction using Transformers and evaluating trading performance using the TradeAI system.

4.1 Prediction Models

The final study aimed to compare the performance of different transformer models on two datasets, which consisted of 3-hourly and daily Bitcoin data. The models' performance was evaluated for short and long-term price prediction using different prediction lengths of 30 seconds, 1 minute, and 3 minutes. To ensure a fair comparison, the input sequence was set at 3 minutes, and the decoder input length was set at 1 minute. This allowed for evaluating all transformers when handling longer data sequences while utilizing the results from the original transformer architecture for trade execution. A longer lookback window would render transformer model results useless, as demonstrated in figure 3.3a.

The hyperparameters for each model were set to their optimal values as seen in table 3.4, and the models were trained for 20 epochs with an early stopping criterion set at a patience level of 3. The models were optimized using the Adam optimizer, with a learning rate of 1×10^{-4}, decaying two times smaller every epoch. The study utilized MLFlow to track the final Mean Squared Error (MSE) and Mean Absolute Error (MAE), which were used to evaluate the performance of each model on each output prediction length.

Table 4.1: Forecasting Results, Sequence Length = 180

Methods	**Metrics**	**3 Hours**			**1 Day**		
		30	60	180	30	60	180
Transformer	MSE	0.074	0.177	0.518	0.044	0.086	0.235
	MAE	0.165	0.263	0.535	0.095	0.153	0.303
Informer	MSE	0.073	0.143	0.507	0.044	0.083	0.238
	MAE	0.157	0.248	0.479	0.095	0.148	0.299
Pyraformer	MSE	**0.053**	**0.115**	**0.373**	**0.042**	**0.081**	**0.226**
	MAE	**0.130**	**0.213**	**0.439**	**0.087**	**0.145**	**0.297**
ARIMA	MSE		0.246			0.523	
	MAE		0.342			0.568	

Table 4.1 provides a comprehensive evaluation of three distinct transformer models, namely the original transformer, Informer transformer, and Pyraformer transformer, with respect to their performance in forecasting Bitcoin data. The findings demonstrate the superior predictive capabilities of transformer models compared to the ARIMA model across various prediction windows. Specifically, transformer models outperformed the ARIMA model in prediction windows of 30 and 60 for the 3-hour dataset and across all prediction lengths for the 1-day dataset. This highlights the ability of transformer models to capture intricate local and global patterns within the data, resulting in more accurate forecasts.

However, it is worth noting that for longer prediction lengths, particularly the 180-minute forecast in the 3-hour dataset, the ARIMA model exhibited better performance compared to the transformer models. This observation emphasises the significance of having large amounts of data for generating reliable long-term predictions. In sparse data, the ARIMA model will be a better predictor. On the other hand, all the transformer models exhibited improved performance as the dataset size increased.

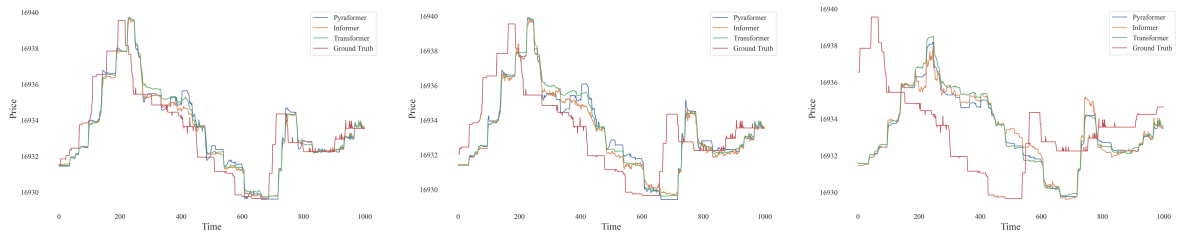
This is an important aspect of transformer models that surpasses Long Short-Term Memory (LSTM) and Recurrent Neural Network (RNN) models since they can handle larger datasets.

The Pyraformer transformer consistently exhibited the highest forecasting accuracy among the transformer models across all prediction lengths. On the 3-hour dataset, it surpassed the original transformer and Informer models by up to 40% for a prediction length of 30 seconds, up to 35% for a prediction length of 1 minute, and up to 27% for a prediction length of 3 minutes. These results suggest that the pyramidal self-attention module effectively captures both local and global dependencies compared to the prob sparse and original self-attention modules.

In the one-day dataset, the performance differences among the three transformer models were relatively marginal. The Pyraformer transformer demonstrated a superiority of approximately 5% over the other models. The Informer transformer model displayed a slight variance of 1-2% in performance, with the transformer model exhibiting better Mean Squared Error (MSE) scores for the 3-minute prediction length. The enhanced performance can be attributed to the positional encoding technique employed in the original transformer architecture, as defined in the Informer model. However, it is noteworthy that with significantly larger look-back windows, such as 5 or 10 minutes, the original transformer model would likely be substantially outperformed by the Informer model, as indicated by the results depicted in Figure 3. This outcome stems from the limitations of the original self-attention module in effectively processing lengthy data sequences.

Overall, the results demonstrate the capability of transformer models to accurately predict cryptocurrency data from order books.

Figure 4.1: Price prediction displayed graphically

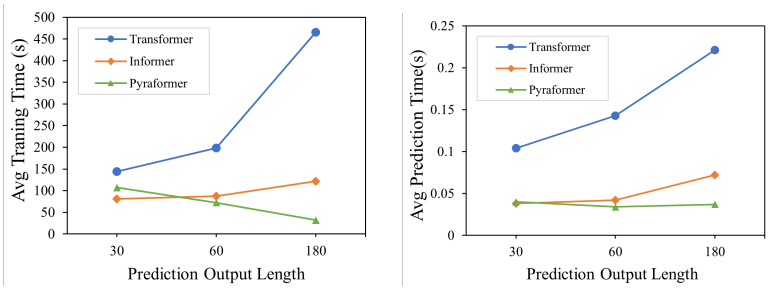


(a) Prediction Length=30 (b) Prediction Length=60 (c) Prediction Length=180

Secondly, a study was conducted to determine how transformer model predictions compare to the real values. Figure 4.2 displays the price prediction results of the transformer models in comparison to the actual values (a larger version of the graphs is available in Appendix G.1). Each graph depicts a small portion of the test set consisting of 1000 data points, equivalent to around 17 minutes of cryptocurrency data. The decision to show a subset of the test set was made to highlight the variations from the actual price patterns. Upon analysis of the results, it was observed that all the models could predict the general pattern of the data with reasonable accuracy when the prediction length was set to 30 seconds or 1 minute. However, the accuracy of the models decreased as the prediction length increased, with all models largely missing the actual values when predicting 180 data points. In some cases, the models could not correctly predict the overall trend of the data, with the actual price trending upwards while the models predicted a downward trend. Additionally, the models tended to underestimate the values during upward price trends and overestimate them during downward trends. This observation highlights the need to examine how this tendency affects the back-testing algorithms.

This study investigated the impact of increasing the prediction output length on the training and inference time of three transformer models, namely the original transformer, Informer transformer, and Pyraformer transformer. Figure 4.2 shows each transformer model's average training and prediction speed over the prediction length. The results demonstrate that the Pyraformer transformer model exhibited a unique behaviour in which the average training time decreased as the prediction output length increased.

Figure 4.2: Average Training Speed and Prediction Speed of Each Transformer



(a) Training Speed (b) Prediction Speed

This indicates that the Pyraformer model training time is not affected by prediction length. However, this is not the case for the original transformer and Informer transformer models, as their average training time increased as the prediction length increased. The original transformer model showed the greatest increase in training time when the prediction length was increased, indicating the limitations of its decoder architecture.

Additionally, figure 4.2b shows that the original transformer model had an average prediction time twice as high as that of the Informer transformer on the 30-point output length, three times as high on the 60-point output length, and four times as high on the 180-point output length, confirming the superiority of the generative-style decoder implemented in the Informer model. The Pyraformer transformer model continued outperforming the other transformers in prediction time, achieving a static time of 0.04s per prediction and confirming its decoder architecture's $O(1)$ time complexity. It is worth noting that while the other transformer models can dynamically change their output length, the Pyraformer transformer model has a fixed prediction output length once trained, and a new model would have to be trained to achieve a different prediction length. This would explain the fact that the training time of Pyraformer is unaffected by the prediction output length. Overall, the findings of this study contribute to a better understanding of the trade-offs between prediction output length and training and inference time in transformer models and can inform the selection of appropriate models for specific applications.

4.2 Trading

The evaluation framework for TradeAI is integrated into the system such that at the end of each market stream simulation, it internally reports: the total net return percentage, the annualised Sharpe ratio (standard metric to compare different investment strategies), the maximum drawdown (the decline in the value of an investment or trading portfolio from its peak value), the maximum drawdown duration, the effective execution time (duration of market stream), the average response time (time from prediction initiation to trade execution) and the number of trades executed.

Numerous variable tuning options are available before backtesting, and we have chosen the following parameters. For Alpha Generation, we employ all three Transformers with the optimal hyperparameters (Table 3.5). We use a prediction length (P) of 30 seconds into the future for all runs, as it demonstrates the best performance. For Order Generation, we execute runs with the Deep Reinforcement Learning model and others with the heuristics-based strategy (threshold T=0.5), and we generally use P as the reload value K, as mentioned in Section 3.4.2. The client's portfolio is initialised with default values for initial capital, maximum BTC position and maximum quantity per trade (100 000 USD, 1, and 0.01 respectively). Finally, 3 distinct 10-minute datasets are used to simulate the market and they correspond to periods within the next 24h from model training; this is to simulate performance in a "train for next day" fashion.

4.2.1 Critical Analysis of the Heuristics-Based Strategy

Table 4.2 showcases the system performance for one of the datasets in accordance with the metrics evaluated (Appendix G.2 for the other 2). Firstly, all runs across the 3 datasets present a neutral/slightly negative net return and Sharpe Ratio. The heuristics-based approach presents a strategy heavily dependent on the price predictions from the transformers. That, and manual observation, lead to the hypothesis that trading underperformance may come directly from the price prediction accuracy not being sufficient for constantly finding the general direction of the market in a profitable magnitude. An alternative hypothesis is that the strategy might not effectively transform the signals for trading purposes. It is possible that more complex methods, such as hybrid approaches combining these predictions with traditional alpha generation techniques or adjusting confidence in a trading decision based on cumulative signal generation, could have been explored for better results. However, we argue that such methods would undermine this strategy´s goal of evaluating the performance of Transformer-based price predictions as a direct source of alpha. Yet another hypothesis is that the chosen simulations do not present enough volatility or simply are not ideal for valuable price prediction as alpha.

Regarding the transformer choice for alpha generation, it is possible to observe that the Pyraformer is by far the fastest model, originating an average response time of 2s (compared with 7 and 20 for the Informer and Transformer, respectively). In addition, it is also the best-performing model in the standalone task of price prediction (Section 3.4). Naturally, it was hypothesized before evaluation that it would generate the best trading strategy and that the longer prediction times for the other 2 models would affect negatively the trading decisions due to slippage. However, in practice, we find the differences negligible, possibly because none of the strategies is polarizing enough to produce a positive Sharpe Ratio. Maximum drawdown and number of trades executed are similar between models.

4.2.2 Critical Analysis of the Deep Reinforcement Learning Strategy

The algorithm is unable to generate a positive net return or Sharpe Ratio. These results raise many questions about possible causes and about the general feasibility of the DRL approach. This subsection theorises on these topics and attempts to give possible answers to these questions.

There are many reasons that might have resulted in a lack of a positive net return. One of them could be the inability of the model to learn when to "trust" the predictions sent by the created predictive models, resulting in inconsistent decisions throughout that decreased profit margin. It also could be the case that the general architecture of the model did not fit the data, despite the research has concluded that the usage of LSTM and Transformer layers can be useful in this setting. Maybe the right parameters and optimisers simply weren't found after hours of experimentation. Furthermore, the fact that the model was trained based on immediate trade execution may have created a wrong representation of market conditions. All of these factors could have contributed different weights to the resulting underperformance. There are two key factors, however, that in hindsight might've contributed more to the results observed.

The choice of technical indicators could have been implemented better. As mentioned within Section 3.5.1, the environment definition plays a big role in the performance of the algorithm. Having replaced the entirety of past and future data with a simple RSI indicator was likely not the wisest approach. In [90] the indicator played a big role in their success, but so did a multitude of other indicators that were used alongside it. Better results might've been observed had these indicators been implemented, as well as any other indicators suitable for this purpose present in the literature.

Secondly, not considering Sentiment Data might also have had an impact. From the research conducted, it is clear that that avenue showed great promise and could have given the lacking information as part of the environment definition that the algorithm required. Unfortunately, as no Sentiment Data could be retrieved for the period of time of the test and train sets, this could not be implemented for this project and this avenue remains unpaved. Future work to explore the usefulness of this data within this

Table 4.2: Trading Strategy Performance Dataset 1 (P:Pyraformer, T:Transformer, I:Informer)

	Heuristics			DRL		
Metric	**P**	**T**	**I**	**P**	**T**	**I**
Net Return (%)	-0.000020	-0.000002	-0.000091	-0.000032	-0.000034	-0.000016
Sharpe Ratio	-0.000069	-0.000006	-0.000447	-0.000453	-0.000399	-0.000131
Max. Drawdown (%)	-0.000135	-0.000118	-0.000104	-0.000032	-0.000034	-0.000052
Max. DD Duration (s)	267s	262s	280s	237s	142s	260s
Exec. Time (s)	364s	364s	364s	364s	364s	364s
Avg. Resp. Time (s)	2.016s	19.356s	6.927s	2.008s	19.932s	7.131s
Num. of Trades	8	8	6	2	2	4

context is required to understand the significance of its omission from the application.

In context, the development of this approach was not a failure. Regardless of not having made a positive net return, its performance is comparable to the heuristic-based strategy and the accuracy of price predictions as alpha signals may be limited.

Chapter 5

Discussion

5.1 Conclusions

This study explored the feasibility of developing a full-stack algorithmic trading system capable of running deep-learning-based trading strategies on Cryptocurrency limit order book data.

Firstly, the study explored novel transformer architectures for high-frequency cryptocurrency price prediction using limit order book data. The original transformer architecture faced challenges in handling time-series data, including processing long sequences and resulting in prolonged inference times. To address these limitations, Informer and Pyraformer time-series transformers were employed, and the original transformer was enhanced with positional encoding methods inspired by Informer. Comparative evaluation against an ARIMA model showed superior performance of all transformer models in capturing intricate patterns, particularly on the daily dataset. However, when the dataset was condensed to a 3-hour timeframe, the ARIMA model outperformed the transformer models in long-term predictions, emphasizing the need for large datasets to capture global dependencies effectively. Notably, although the performance gap between the Informer model and the improved original transformer was minimal, the Informer model demonstrated greater efficacy in handling long data sequences, as evident from its training and inference times. Moreover, it should be noted that if longer input sequences were chosen, the transformer model would face challenges in achieving comparable results due to the inherent quadratic complexity of the self-attention module. Among the transformer models, the Pyraformer model exhibited the best performance across both datasets, showcasing superior predictive accuracy and shorter training and inference times, thus positioning it as a promising choice for real-time applications and production settings.

Importantly, this study contributes to understanding the effectiveness of transformer models in predicting medium-frequency cryptocurrency prices, highlighting the need for further research in this field. The research on cryptocurrency price prediction lags behind that of stock market forecasting, with both domains underutilising time-series transformers. To address this gap, this project aims to accelerate research into cryptocurrency price forecasting by leveraging innovative time-series transformers. Shifting the research focus from stock market forecasting to cryptocurrency prediction offers researchers advantages, as historical cryptocurrency data is more readily accessible compared to historical stock market data. This shift also opens avenues for novel research in finance and economics.

Furthermore, this research introduces a novel framework for formulating DRL strategies, integrating the results from predictive models into its environment definition. Several different structures were weighed and tested, having settled for a finished DRL utilizing the Trade Completion reward function and the A2C architecture. Although the final product was unable to generate profit, this algorithm still performed on par with heuristic-based strategies - which are promising results that should be explored with further research.

We have successfully and innovatively combined an algorithmic trading system capable of backtesting with deep learning methods and with a microservices architecture. TradeAI is capable of accommodating deep learning methods in 2 distinct components of its structure: alpha generation and order generation. It trades on a frequency ranging between 1-30s and it reports them to the respective client. Regarding the feasibility of trading using Transformer-based price predictions as alpha signals that are then processed by a DRL for order generation, we obtain marginally negative results for Sharpe Ratio and total net return percentage. The cause(s) for such results is unclear, though we hypothesize it being the accuracy of using

the price predictions as alpha. Nevertheless, we remain optimistic the developed trading framework has potential for tuning and creating successful trading strategies. In some ways, this is the end of the beginning.

5.2 Ideas for future work

In the realm of future work, several key areas can be explored to further enhance the performance and capabilities of algorithmic trading systems. This section outlines the potential directions for future research.

In order to delve deeper into the influence of dataset size on the performance of transformer models, future research could direct its focus toward employing even larger datasets spanning durations of 3 or 7 days. This investigation would provide insights into the scalability of transformer models, as well as confirm whether their performance continues to improve proportionally with increasing dataset sizes. Additionally, it would be of great interest to explore the effectiveness of time-series transformers in predicting both high-frequency data and long-frequency data. Investigating the performance of these transformers in handling such diverse temporal characteristics would shed light on their ability to capture short-term and long-term predictions and indicate which one they operate best on.

Furthermore, conducting a comprehensive analysis between transformer models and state-of-the-art LSTM models would be insightful. Such an analysis would provide an understanding of the respective strengths and weaknesses inherent in both models such as prediction accuracy, computational efficiency, and adeptness in managing long-term dependencies. This evaluation would enable researchers to make informed decisions when selecting the most appropriate model for specific prediction tasks of Cryptocurrency data.

Moreover, as part of future work in the domain of Deep Reinforcement Learning, there is potential for implementing sentiment analysis techniques to augment the forecasting models. By integrating sentiment analysis, the models can leverage additional contextual information derived from social media, news articles, or other relevant sources, which may contribute to more accurate decision-making in algorithmic trading. In addition, exploring more complex neural network architectures holds promise for advancing the capabilities of the decisions. Experimentation with different network structures, such as convolutional neural networks (CNNs), recurrent neural networks (RNNs), or their hybrid combinations, can uncover novel approaches to effectively capture temporal patterns and dependencies within the time-series data. The obtained results are promising, but proper further exploration might be exactly what is missing to return profitability.

Lastly, the logical progression of the algorithmic trading system involves two distinct aspects. Firstly, a remarkable framework is in place for developing trading strategies based on deep learning, and further research in this area is highly encouraged. It is firmly believed that there remains a multitude of untapped possibilities within the system, holding the potential for creating profitable trading strategies. Moreover, there is room for enhancing the market simulation by directly utilizing unprocessed order book data, thus improving the accuracy and realism of the trading environment. Additionally, efforts can be directed towards readying the TradeAI system for live trading by effectively handling real-time market data feeds and optimizing the execution module for seamless trade execution. On the other hand, the TradeAI system has the potential to evolve into an algorithmic-trading-as-a-service paradigm, particularly if it progresses toward an MLOps level 2 [48] classification and establishes suitable microservice scalability rules and optimal resource parameters for the server infrastructure.

References

[1] *Binance API*. [Accessed: 27 November 2022]. Available from: https://www.binance.com/en/binance-api.

[2] *Binance API Terms of Use*. [Online]. 2022. [Accessed: 27 November 2022]. Available from: https://www.binance.com/en/terms.

[3] *Binance Order Book*. [Online]. 2022. [Accessed: 27 November 2022]. Available from: https://www.binance.com/en/orderbook/BTC_USDT.

[4] *Bitcoin crashes as Elon Musk announces Tesla cars can no longer be bought with cryptocurrency*. [Online]. 2023. [Accessed: 27 February 2023]. Available from: https://www.independent.co.uk/space/elon-musk-bitcoin-crash-tesla-cryptocurrency-b1846650.html.

[5] *CC0 licence*. [Online]. 2022. [Accessed: 27 November 2022]. Available from: https://creativecommons.org/publicdomain/zero/1.0/.

[6] *Coinbase*. [Online]. 2022. [Accessed: 27 November 2022]. Available from: https://www.coinbase.com/explore.

[7] *CoinMarketCap*. [Online]. 2022. [Accessed: 27 November 2022]. Available from: https://coinmarketcap.com/.

[8] *Crypto Market Sentiment Analysis - Stockgeist*. [Online]. 2023. [Accessed: 27 February 2023]. Available from: https://www.stockgeist.ai/crypto-sentiment-analysis/.

[9] *DVC*. [Software]. 2022. [Accessed: 27 November 2022]. Available from: https://dvc.org/.

[10] *El Salvador's adoption of bitcoin as legal tender is pure folly*. [Online]. 2023. [Accessed: 10 May 2023]. Available from: https://www.theguardian.com/business/2021/sep/24/el-salvador-adoption-of-bitcoin-as-legal-tender-is-pure-folly.

[11] *Kaggle*. [Online]. 2022. [Accessed: 27 November 2022]. Available from: https://www.kaggle.com/.

[12] *Proximal Policy Optimization - Open AI*. [Online]. 2023. [Accessed: 2 May 2023]. Available from: https://spinningup.openai.com/en/latest/algorithms/ppo.html.

[13] *UK cryptocurrency tax guide: everything you need to know*. [Online]. 2023. [Accessed: 10 May 2023]. Available from: https://www.unbiased.co.uk/discover/personal-finance/savings-investing/uk-cryptocurrency-tax-guide-everything-you-need-to-know.

[14] *What's the Environmental Impact of Cryptocurrency?* [Online]. 2023. [Accessed: 10 May 2023]. Available from: https://www.investopedia.com/tech/whats-environmental-impact-cryptocurrency/.

[15] C. R. Abhijit Sharang. Using machine learning for medium frequency derivative portfolio trading. 2015. [Accessed 27 November 2022]. Available from: https://arxiv.org/pdf/1512.06228.pdf.

[16] E. M. Aistis Raudys, Vaidotas Lenčiauskas. Moving averages for financial data smoothing. 2013. [Accessed 27 November 2022]. Available from: https://www.forexfactory.com/attachment/file/3794157?d=1605879155.

[17] C. Alexander and M. Dakos. A critical investigation of cryptocurrency data and analysis. *Quantitative Finance*, 20(2):173–188, 2020.

[18] R. Almgren and N. Chriss. Optimal execution of portfolio transactions. 2000. [Accessed 27 November 2022]. Available from: https://www.smallake.kr/wp-content/uploads/2016/03/optliq.pdf.

[19] M. A. Ammer and T. H. H. Aldhyani. Deep learning algorithm to predict cryptocurrency fluctuation prices: Increasing investment awareness. *Electronics*, 11(15), 2022.

[20] M. S. ANDREI KIRILENKO, ALBERT S. KYLE and T. TUZUN. The flash crash: High-frequency trading in an electronic market. 2017. [Accessed 27 November 2022]. Available from: https://doi.org/10.1111/jofi.12498.

[21] M. A. Andrew Boutros, Brett Grady and P. Chow. Build fast, trade fast: Fpga-based high-frequency trading using high-level synthesis. 2020. [Accessed 27 November 2022]. Available from: https://ieeexplore.ieee.org/abstract/document/8279781.

[22] R. Arbi. A reproducible approach to equity backtesting. 2019. [Accessed 27 November 2022]. Available from: https://open.uct.ac.za/bitstream/handle/11427/31158/thesis_sci_2019_arbi_riaz.pdf?sequence=1&isA

[23] D. Bahdanau, K. H. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 9 2014.

[24] F. Becattini and T. Uricchio. Memory networks. *MM 2022 - Proceedings of the 30th ACM International Conference on Multimedia*, pages 7380–7382, 10 2014.

[25] K. H. . B. B. . J. Beda. *Kubernetes: Up and Running: Dive into the Future of Infrastructure.* 2017. [Accessed 27 November 2022]. Available from: https://www.pdfdrive.com/kubernetes-up-and-running-dive-into-the-future-of-infrastructure-e185935499.html.

[26] M. Z. e. a. Benjamin Hindman, Andy Konwinski. Mesos: A platform for fine-grained resource sharing in the data center. 2011. [Accessed 27 November 2022]. Available from: https://people.eecs.berkeley.edu/ alig/papers/mesos.pdf.

[27] H. N. Bhandari, B. Rimal, N. R. Pokhrel, R. Rimal, K. R. Dahal, and R. K. Khatri. Predicting stock market index using lstm. *Machine Learning with Applications*, 9:100320, 9 2022.

[28] Binance. Trading fees, 2023. [Accessed 20 April 2023]. Available from: https://www.binance.com/en/fee/schedule.

[29] D. S. Brendan Burns, Eddie Villalba and L. Evenson. *Kubernetes Best Practices: Blueprints for Building Successful Applications on Kubernetes.* 2019. [Accessed 27 November 2022]. Available from: https://dokumen.pub/kubernetes-best-practices-blueprints-for-building-successful-applications-on-kubernetes-1492056472-9781492056478.html.

[30] L. H. P. Brian Hurst, Yao Hua Ooi. A century of evidence on trend-following investing. 2017. [Accessed 27 November 2022]. Available from: https://fairmodel.econ.yale.edu/ec439/hurst.pdf.

[31] C. Cao, O. Hansch, and X. Beardsley. The information content of an open limit-order book. *Journal of Futures Markets*, 29:16 – 41, 01 2009.

[32] Y. G. Cinar, H. Mirisaee, P. Goswami, E. Gaussier, A. Aït-Bachir, and V. Strijov. Position-based content attention for time series forecasting with sequence-to-sequence rnns. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10638 LNCS:533–544, 2017.

[33] Containerd. Containerd, 2022. [Accessed 27 November 2022]. Available from: (https://containerd.io/.

[34] CRI-O. Cri-o, 2022. [Accessed 27 November 2022]. Available from: https://cri-o.io/.

[35] D. G. e. a. D. Sculley, Gary Holt. Machine learning: The high-interest credit card of technical debt. 2022. [Accessed 27 November 2022]. Available from: https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/43146.pdf.

[36] M. M. L. D. P. DAVID EASLEY and M. O'HARA. The microstructure of the 'flash crash': Flow toxicity, liquidity crashes, and the probability of informed trading. 2011. [Accessed 27 November 2022]. Available from: https://www.semanticscholar.org/paper/The-Microstructure-of-the-%E2%80%9CFlash-Crash%E2%80%9D%3A-Flow-and-Easley-Prado/6bf500d5b533ce28c3a9ca8532e0701ddb482f9d.

[37] M. A. H. Dempster and V. Leemans. An automated fx trading system using adaptive reinforcement learning, 2004.

[38] A. Detzel, H. Liu, J. Strauss, G. Zhou, and Y. Zhu. Learning and predictability via technical analysis: Evidence from bitcoin and stocks with hard-to-value fundamentals. *Financial Management*, 50:107–137, 3 2021.

[39] Docker. Swarm mode overview, 2022. [Accessed 27 November 2022]. Available from: https://docs.docker.com/engine/swarm/.

[40] S. H. Dominik Kreuzberger, Niklas Kühl. Machine learning operations (mlops): Overview, definition, and architecture. 2022. [Accessed 27 November 2022]. Available from: https://arxiv.org/abs/2205.02302.

[41] D. Fister, M. Perc, and T. Jagrič. Two robust long short-term memory frameworks for trading stocks. *Applied Intelligence*, 51:7177–7195, 10 2021.

[42] V. G. *Pairs trading: Quantitative methods and analysis.* 2004. [Accessed 27 November 2022]. Available from: https://www.booksfree.org/pairs-trading-quantitative-methods-and-analysis-by-ganapathy-vidyamurthy-pdf/.

[43] D. Garcia, C. J. Tessone, P. Mavrodiev, and N. Perony. The digital traces of bubbles: feedback cycles between socio-economic signals in the bitcoin economy. *Journal of The Royal Society Interface*, 11(99):20140623, 2014.

[44] N. Garg. *Apache Kafka.* 2013. [Accessed 27 November 2022]. Available from: https://www.pdfdrive.com/apache-kafka-set-up-apache-kafka-clusters-and-develop-custom-message-producers-and-consumers-using-practical-hands-on-examples-e185785013.html.

[45] R. K. G. Gatev E., Goetzmann W. N. Pairs trading: Performance of a relative-value arbitrage rule. 2006. [Accessed 27 November 2022]. Available from: http://stat.wharton.upenn.edu/ steele/Courses/434/434Context/PairsTrading/PairsTradingGGR.pdf.

[46] H. GB and S. N. B. Cryptocurrency price prediction using twitter sentiment analysis. pages 13–22, 3 2023.

[47] P. T. Giuseppe Nuti, Mahnoosh Mirghaemi and C. Yingsaeree. Algorithmic trading. 2011. [Accessed 27 November 2022]. Available from: https://ieeexplore.ieee.org/document/5696713.

[48] Google. Mlops: Continuous delivery and automation pipelines in machine learning, 2023. [Accessed 13 March 2023]. Available from: https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning.

[49] M. Greco, M. Spagnoletta, A. Appice, and D. Malerba. Applying machine learning to predict closing prices in stock market: A case study. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 12591 LNAI:32–39, 2021.

[50] R. L. Guangliang He. The intuiton behind black-litterman model portfolios. 2002. [Accessed 27 November 2022]. Available from: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=334304.

[51] H. Guo, X. Lin, J. Yang, Y. Zhuang, J. Bai, T. Zheng, L. Zheng, W. Hou, B. Zhang, and Z. Li. Translog: A unified transformer-based framework for log anomaly detection. 12 2021.

[52] B. E. haddaoui, R. Chiheb, R. Faizi, and A. E. Afia. The influence of social media on cryptocurrency price: A sentiment analysis approach. *International Journal of Computing and Digital Systems*, 13:1–15, 3 2023.

[53] M. L. Halls-Moore. Optimal execution of portfolio transactions. 2015. [Accessed 27 November 2022]. Available from: https://raw.githubusercontent.com/englianhu/binary.com-interview-question/fcad2844d7f10c486f3601af9932f49973548e4b/reference/Successful%20Algorithmic%20Trading.pdf.

[54] M. L. Halls-Moore. Optimal execution of portfolio transactions. 2016. [Accessed 27 November 2022]. Available from: https://www.pdfdrive.com/advanced-algorithmic-trading-e184792980.html.

[55] S. Hansun and J. C. Young. Predicting lq45 financial sector indices using rnn-lstm. *Journal of Big Data*, 8:1–13, 12 2021.

[56] M. Hilbert and D. Darmon. How complexity and uncertainty grew with algorithmic trading. 2020. [Accessed 27 November 2022]. Available from: https://www.mdpi.com/1099-4300/22/5/499.

[57] B. Huang, Y. Huan, L. D. Xu, L. Zheng, and Z. Zou. Automated trading systems statistical and machine learning methods and hardware implementation: a survey. *https://doi.org/10.1080/17517575.2018.1493145*, 13:132–144, 1 2018.

[58] K. C. e. a. Ioana Baldini, Paul Castro. Serverless computing: Current trends and open problems. 2017. [Accessed 27 November 2022]. Available from: https://arxiv.org/abs/1706.03178.

[59] T. H. Jonathan Brogaard and R. Riordan. High-frequency trading and price discovery. 2013. [Accessed 27 November 2022]. Available from: https://www.ecb.europa.eu/pub/pdf/scpwps/ecbwp1602.pdf.

[60] T. Kabbani and E. Duman. Deep reinforcement learning approach for trading automation in the stock market. 7 2022.

[61] D. K. Kasun Indrasiri. *gRPC: Up and Running*. 2020. [Accessed 27 November 2022]. Available from: https://dokumen.pub/grpc-up-amp-running-building-cloud-native-applications-with-go-and-java-for-docker-and-kubernetes-9781492058335-b-6485645.html.

[62] I. Katib, R. Mehmood, and N. Malibari. Predicting stock closing prices in emerging markets with transformer neural networks: The saudi stock exchange case. *Article in International Journal of Advanced Computer Science and Applications*, 12:2021, 2021. To the athurs it was the first time the transformers were used for the stock markter prediction.

[63] A. N. Kercheval and Y. Zhang. Modelling high-frequency limit order book dynamics with support vector machines. *Quantitative Finance*, 15(8):1315–1329, 2015.

[64] D. S. Khalid Salama, Jarek Kazmierczak. Practitioners guide to mlops: A framework for continuous delivery and automation of machine learning, 2021. [Accessed 27 November 2022]. Available from: https://cloud.google.com/architecture/mlops-continuous-delivery-and-automation-pipelines-in-machine-learning.

[65] A. M. Khedr, I. Arif, P. R. P V, M. El-Bannany, S. M. Alhashmi, and M. Sreedharan. Cryptocurrency price prediction using traditional statistical and machine-learning techniques: A survey. *Intelligent Systems in Accounting, Finance and Management*, 28(1):3–34, 2021.

[66] O. Kuchaiev and B. Ginsburg. Factorization tricks for lstm networks. *5th International Conference on Learning Representations, ICLR 2017 - Workshop Track Proceedings*, 3 2017.

[67] Z. Lanbouri and S. Achchab. Stock market prediction on high frequency data using long-short term memory. *Procedia Computer Science*, 175:603–608, 1 2020.

[68] P. Lara-Benítez, L. Gallego-Ledesma, M. Carranza-García, and J. M. Luna-Romera. Evaluation of the transformer architecture for univariate time series forecasting. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 12882 LNAI:106–115, 2021.

[69] A. Leccese. Machine learning in finance: Why you should not use lstm's to predict the stock market - bluesky capital. 1 2019.

[70] S. R. Leonard Richardson. *RESTful web services*. 2007. [Accessed 27 November 2022]. Available from: https://www.crummy.com/writing/RESTful-Web-Services/RESTful_Web_Services.pdf.

[71] J. H. Li Erran Li, Eric Chen, P. Zhang, and L. Wang. Scaling machine learning as a service. 2017. [Accessed 27 November 2022]. Available from: https://proceedings.mlr.press/v67/li17a.html.

[72] K. Lien. *Day Trading and Swing Trading the Currency Market: Technical and Fundamental Strategies to Profit from Market Moves*. 2009. [Accessed 27 November 2022]. Available from: https://www.joesenforex.com/wp-content/uploads/2014/03/Day-Trading.pdf.

[73] J.-H. L. Marco Avellaneda. Statistical arbitrage in the u.s. equities market. 2008. [Accessed 27 November 2022]. Available from: https://math.nyu.edu/ avellane/AvellanedaLeeStatArb071108.pdf.

[74] J. B. Meenu Mary John, Helena Holmström Olsson. Towards mlops: A framework and maturity model. 2021. [Accessed 27 November 2022]. Available from: https://ieeexplore.ieee.org/document/9582569.

[75] I. F. . A. Melnikov. The websocket protocol. 2011. [Accessed 27 November 2022]. Available from: https://greenbytes.de/tech/webdav/rfc6455.pdf.

[76] A. J. Menkveld. High frequency trading and the new market makers. 2013. [Accessed 27 November 2022]. Available from: https://www.sciencedirect.com/science/article/pii/S1386418113000281.

[77] D. Merkel. Docker: Lightweight linux containers for consistent development and deployment, 2014. [Accessed 27 November 2022]. Available from: https://dl.acm.org/doi/fullHtml/10.5555/2600239.2600241.

[78] T. Muhammad, A. B. Aftab, M. M. Ahsan, M. M. Muhu, M. Ibrahim, S. I. Khan, and M. S. Alam. Transformer-based deep learning model for stock price prediction: A case study on bangladesh stock market. 2022. Small refrence for the use of transformers for stock market.

[79] R. K. Narang. *Inside the Black Box: A Simple Guide to Quantitative and Algorithmic Trading*. 2013. [Accessed 27 November 2022]. Available from: https://usermanual.wiki/Pdf/Rishi20K20Narang20Inside20the20black20box20a20simple 20guide20to20quantitative20and20highfrequency20trading.1234062944.pdf.

[80] S. Newman. *Building Microservices: Designing Fine-Grained Systems*. 2015. [Accessed 27 November 2022]. Available from: https://dokumen.pub/building-microservices-designing-fine-grained-systems.html.

[81] M. S. Nielsen. High frequency crypto limit order book data. [Online]. 2022. [Accessed 27 November 2022]. Available from: https://www.kaggle.com/datasets/martinsn/high-frequency-crypto-limit-order-book-data.

[82] Nkaz001. Hftbacktest, 2023. [Accessed 12 March 2023]. Available from: https://hftbacktest.readthedocs.io/en/latest/.

[83] A. Ntakaris, M. Magris, J. Kanniainen, M. Gabbouj, and A. Iosifidis. Benchmark dataset for mid-price forecasting of limit order book data with machine learning methods. *Journal of Forecasting*, 37, 08 2018.

[84] C. Pahl. Containerization and the paas cloud. 2015. [Accessed 27 November 2022]. Available from: https://ieeexplore.ieee.org/document/7158965.

[85] F. C. L. Paiva, L. K. Felizardo, R. A. da Costa Bianchi, and A. H. R. Costa. Intelligent trading systems: A sentiment-aware reinforcement learning approach. 11 2021.

[86] B. Pisani. Flash crash' 5 years later: What have we learned?, 2015. [Accessed 27 November 2022]. Available from: https://www.cnbc.com/2015/05/05/flash-crash-5-years-later-what-have-we-learned.html.

[87] E. S. Ponomarev, I. V. Oseledets, and A. S. Cichocki. Using reinforcement learning in the algorithmic trading problem. *Journal of Communications Technology and Electronics*, 64:1450–1457, 12 2019.

[88] J. Qiu, B. Wang, and C. Zhou. Forecasting stock prices with long-short term memory neural network based on attention mechanism. 2020.

[89] R. T. Rama Cont, Sasha Stoikov. A stochastic model for order book dynamics. 2010. [Accessed 27 November 2022]. Available from: http://rama.cont.perso.math.cnrs.fr/pdf/CST2010.pdf.

[90] J. Sadighian. Deep reinforcement learning in cryptocurrency market making. 11 2019.

[91] H. Sak, A. W. Senior, and F. Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *INTERSPEECH*, pages 338–342, 2014.

[92] D. Salinas, V. Flunkert, J. Gasthaus, and T. Januschowski. Pyraformer: Low-complexity pyramidal attention for long-range time series modeling and forecasting. *International Journal of Forecasting*, 36:1181–1191, 1 2022.

[93] T. B. Samuel Idowu, Daniel Strüber. Asset management in machine learning: A survey. 2021. [Accessed 27 November 2022]. Available from: https://arxiv.org/abs/2102.06919.

[94] E. L. Sasu Mäkinen, Henrik Skogström and T. Mikkonen. Who needs mlops: What data scientists seek to accomplish and how can mlops help? 2021. [Accessed 27 November 2022]. Available from: https://arxiv.org/abs/2103.08942.

[95] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *5th International Conference on Learning Representations, ICLR 2017 - Conference Track Proceedings*, 1 2017.

[96] J. M. H. Shreya Shankar, Rolando Garcia and A. G. Parameswaran. Operationalizing machine learning: An interview study. 2022. [Accessed 27 November 2022]. Available from: https://arxiv.org/abs/2209.09125.

[97] R. C. Staudemeyer and E. R. Morris. Understanding lstm-a tutorial into long short-term memory recurrent neural networks. *arXiv preprint arXiv:1909.09586*, 2019.

[98] M. E. F. Stephen Soltesz, Herbert Pötzl, A. Bavier, and L. Peterson. Container-based operating system virtualization: A scalable, high-performance alternative to hypervisors. 2007. [Accessed 27 November 2022]. Available from: https://zaoxing.github.io/files/course/EuroSys07_Container.pdf.

[99] Q. Strategies. What percentage of trading is algorithmic?, 2023. [Accessed 20 April 2023]. Available from: https://www.quantifiedstrategies.com/what-percentage-of-trading-is-algorithmic/.

[100] C. M. J. TERRENCE HENDERSHOTT and A. J. MENKVELD. Does algorithmic trading improve liquidity? 2011. [Accessed 27 November 2022]. Available from: https://faculty.haas.berkeley.edu/hender/Algo.pdf.

[101] T. Théate and D. Ernst. An application of deep reinforcement learning to algorithmic trading. 4 2020.

[102] A. Tsantekidis, N. Passalis, A. Tefas, J. Kanniainen, M. Gabbouj, and A. Iosifidis. Using deep learning to detect price change indications in financial markets. 2017.

[103] A. Tsantekidis, N. Passalis, A. Tefas, J. Kanniainen, M. Gabbouj, and A. Iosifidis. Using deep learning for price prediction by exploiting stationary limit order book features. *Applied Soft Computing Journal*, 93, 10 2018.

[104] Y. Tu. *Predicting high-frequency stock market by neural networks*. PhD thesis, MS thesis, Dept. Mathematics, Imperial College London, London, United Kingdom, 2020.

[105] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Łukasz Kaiser, and I. Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017-December:5999–6009, 6 2017.

[106] C. Wang, Y. Chen, S. Zhang, and Q. Zhang. Stock market index prediction using deep transformer model. *Expert Systems with Applications*, 208:118128, 12 2022.

[107] J. Wang, Q. Cui, X. Sun, and M. He. Asian stock markets closing index forecast based on secondary decomposition, multi-factor analysis and attention-based lstm model. *Engineering Applications of Artificial Intelligence*, 113:104908, 8 2022.

[108] Q. Wen, T. Zhou, C. Zhang, W. Chen, Z. Ma, J. Yan, and L. Sun. Transformers in time series: A survey. 2 2022. In section 2) paper disscusses and gives implementation details of the vanilla transformer.

[109] Y. L. Xing Sheng, Shuangshuang Hu. The micro-service architecture design research of financial trading system based on domain engineering. 2018. [Accessed 27 November 2022]. Available from: https://www.atlantis-press.com/proceedings/ssmi-18/55913117.

[110] Z. Z. Yang Li, Wanshan Zheng. Deep robust reinforcement learning for practical algorithmic trading. 2019. [Accessed 27 November 2022]. Available from: https://ieeexplore.ieee.org/document/8786132.

[111] I. Zaznov, J. Kunkel, A. Dufour, and A. Badii. Predicting stock price changes based on the limit order book: A survey. *Mathematics*, 10, 2022.

[112] A. Zeng, M. Chen, L. Zhang, and Q. Xu. Are transformers effective for time series forecasting? 5 2022.

[113] H. Zhou, S. Zhang, J. Peng, S. Zhang, J. Li, H. Xiong, and W. Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35:11106–11115, 5 2021.

[114] S. A. Zineb Lanbouri. Stock market prediction on high frequency data using long-short-term memory. 2020. [Accessed 27 November 2022]. Available from: https://www.sciencedirect.com/science/article/pii/S1877050920317877.

[115] Z. Zou and Z. Qu. Using lstm in stock prediction and quantitative trading. *CS230: Deep Learning, Winter*, pages 1–6, 2020.

Appendices

Appendix A

Ethical, Legal and Other issues

A.1 Cryptocurrency effects on the environment

It is no secret that mining Cryptocurrency is an energy-dense process [14]. And due to our current energy climate, much of the electricity required for it is obtained through the burning of fossil fuels [14]. As mentioned in [14] about Bitcoin in particular, it "computes to around 1,455.8 kilowatt-hours of electricity per transaction, the same amount of power consumed by the average American household over 49.9 days".

Due to the magnitude of buy and sell orders placed by the program developed as part of this report, the contribution of an application like this to the environment can be quite dire. There is not much that can be done directly in terms of the architecture to mitigate this effect. One approach could be to incentivise the trading strategies to perform as few transactions as possible, but this could impact the performance of the algorithms.

The best course of action would be to select coins that are not as energy-intensive to mine (such as Ethereum, for example [14]), and invest in greener sources of energy. The latter, however, is outside of the sphere of influence of the writers of this report and, as a result, of the scope of the project.

A.2 Legal taxation of Bitcoin income

Depending on the location of a client in the world, they might be required to declare their earnings of Crytopcurrency. The laws and regulations surrounding this are often vague and confusing, with some countries not recognising Crytocurrency as anything taxable, others (like the UK) as assets such as stocks [13], and some even adopting Bitcoin as a legal tender (like El Salvador [10]).

The analysis performed by the application within this report doesn't take any of these factors into account, nor does the application itself. Therefore, extra work would have to be done in order to legally deploy and analyse the feasibility of the application in different parts of the world

A.3 Data ethics

Referenced Kaggle data sources are available under the CC0: Public Domain license [5] that allows the use of data for any purpose without restriction. Binance terms of use allow the use of Binance Services, therefore Binance API, to be used for individual, non-commercial purposes [2].

Appendix B

Project Management

B.1 Methodology

The project is approached with agile methodology by performing iterative development and executing weekly sprints. In addition, MLOps methodology is utilised to assign team roles. One team member is responsible for Data in general, the ML team is composed by 3 people, another team member is responsible for Operations and the last member is responsible for developing a client interface.

B.2 Tasks, milestones and timeline

The timeline includes 3 iterations which should total in 10 sprints.

- The first iteration should implement the project end-to-end and achieve the most basic features – 3 sprints
- The second iteration should extend functionality by implementing multiple trading strategies and fully functional UI – 4 sprints
- The third iteration should improve the project on the existing features ad explore new possible ones – 3 sprints

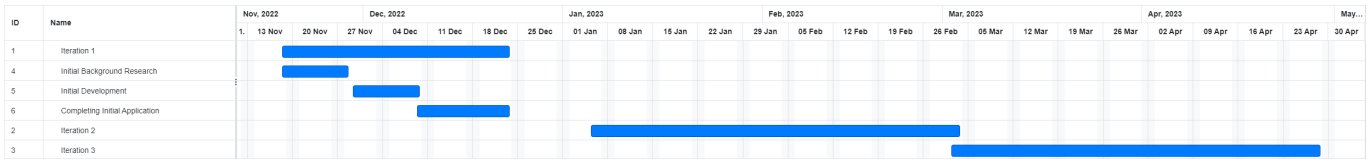


Figure B.1: Development Timeline

Appendix C

Extra Literature Review

C.1 Attention Models

Attention mechanisms have emerged as a breakthrough in sequence modelling, addressing the limitations of traditional RNNs and LSTMs in processing long data sequences. By enabling the network to selectively focus on the most relevant parts of the input sequence, attention models have revolutionized the field of natural language processing [23].

Research has shown that attention mechanisms are not limited to natural language processing but can also improve the performance of time-series forecasting models. Incorporating Position-Based Content Attention in RNNs [32] has been found to improve the detection of pseudo-periods in time series data [24]. By utilizing attention to identify and reuse time intervals with strong correlations, these models achieve superior performance in time series forecasting, largely outperforming traditional LSTM and RNN models, including stock market data. Building on this success, attention-based LSTMs have become popular in stock price prediction. One study combined a wavelet transform with an attention-based LSTM model to forecast stock prices with high accuracy on S&P 500 and DJIA datasets [88]. Another study proposed a hybrid model based on secondary decomposition, multi-factor analysis, and attention-based LSTM to predict the closing index of Asian stock markets with mean average percentage errors below 1% [107]. Using attention-based LSTMS and RNN models for time-series forecasting has opened up new ways to improve existing models' performance and has been shown to largely outperform traditional models.

However, the architecture of LSTMs and RNNs still presents challenges that limit their scalability to large amounts of data. Specifically, their sequential computation hinders parallelization, making it difficult to train them on large datasets such as order book data [105]. Despite efforts to address these limitations through factorization tricks [66] and conditional computation [95], the fundamental challenge of sequential computation remains. This has led to the development of transformer models [105], which have shown great promise in processing long sequences of data while being highly parallelizable, making them well-suited for training on large datasets like order book data. As a result, Transformer models have provided a solution to the scalability issues recurrent models face, rendering them a promising technique for training on large datasets.

C.2 Level 2 MLOps Pipeline

C.3 Applications as Microservices

Microservices architecture is best known for use cases where a complex system can be divided into smaller, independent applications – usually by splitting it into containers. By encapsulating each application and its dependencies, the development, deployment, scaling, and updating of individual components can be achieved without affecting the entire system [80].

C.4 Containers

Containers have emerged as a significant innovation in software development and deployment, offering a lightweight and efficient alternative to traditional virtualization technologies [98]. They provide an ideal solution for continuous integration and delivery (e.g., GitLab CI/CD) due to their ease of building,

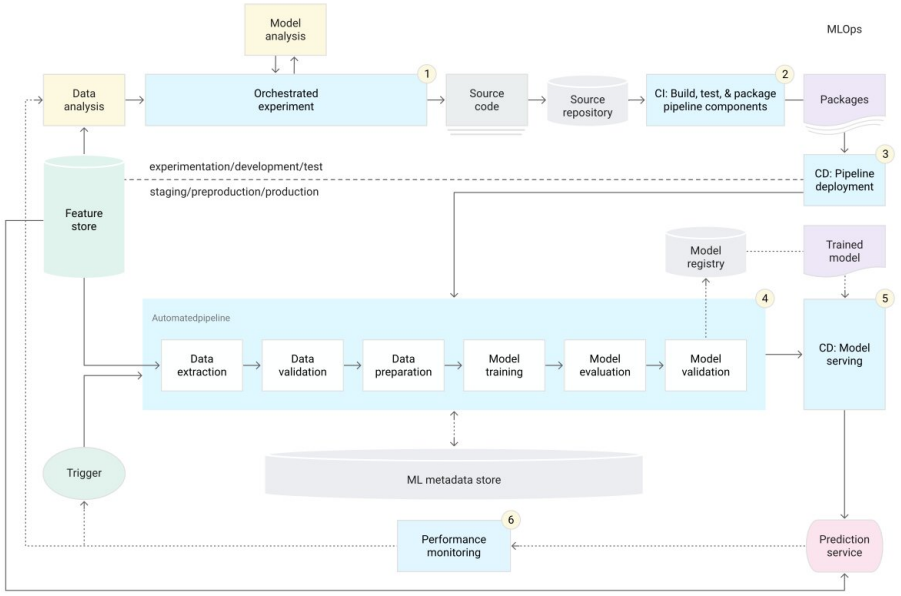


Figure C.1: Level 2 MLOps Pipeline [48]

testing, and deployment. Positioned between virtual machines (VM) and function-as-a-service on the spectrum of infrastructure control and isolation, containers leverage operating system-level virtualization to isolate an application and its dependencies from the underlying system, while potentially sharing the operating system kernel.

Container runtimes are responsible for creating, running, and managing containers on a host system, providing isolation and resource allocation to each container [84]. There are various, the most widely adopted being Docker. It introduced the concept of container images and a standardized format for packaging and distributing containerized applications, becoming the foundation for deploying microservices [77]. In addition, Docker provides not only runtime but other container management tools such as Docker Compose for inter-container communication and I/O operations with external files ("volumes"). Alternatives are CRI-O [34], a container runtime designed for Kubernetes with a focus on security and performance, and Containerd [33], with a focus on more modular and lower-level container runtimes.

As one of many successful examples, Y. L. Xing Sheng & Shuangshuang Hu (2018) [109] find that the Interbank RMB Trading System had issues with high complexity, deployment difficulty and limited expansion capability – a classic legacy system. Thus, they propose a requirements-capture based on domain engineering and a gradual transfer to a new microservice-based architecture that solves them.

C.5 Containers on Cloud Platforms

Container orchestration is a crucial aspect of deploying containerized applications, It automates the deployment, management, scaling, and networking of containers. In recent times, cloud platforms have provided services that facilitate the use of container orchestration tools.

Kubernetes is the most widely used container orchestration platform and it provides a rich set of features, such as horizontal scaling, rolling updates, self-healing, and load balancing, making it an ideal choice for managing complex, large-scale containerized applications [25]. Alternatives are Docker Swarm [39], which is simple and suitable for small-scale applications, and Apache Mesos [26], which abstracts the resources of an entire data centre into a single pool, allowing for efficient and fine-grained resource allocation.

Established Cloud Platforms have adopted Kubernetes as the de facto standard [25]. Amazon Web

Services (AWS), Google Cloud Platform (GCP), and Microsoft Azure have developed managed Kubernetes services (EKS, GKE, and AKS, respectively) with both command line and simplified UI interfaces. These have the advantage of integration with other useful services like File Storage and Container Registry and naturally take advantage of seamless resource scaling.

The concept of Serverless Containers or Container-as-a-Services has been introduced by cloud providers in a push for the simplification of container deployment and scalability. A developer still has the flexibility of defining container resource requirements and scalability rules, but there's no need to manage the underlying infrastructure [58]. AWS Fargate, Google Cloud Run, and Azure Container Instances (ACI) are the best-known Container-as-a-Service solutions.

In the context of containerized applications, continuous deployment strategies must address unique challenges associated with containers, such as versioning, and updating. Versioning can be managed using git-like solutions to keep track of the latest image (eg. Docker Hub and Azure Container Registry). The updating mechanism can be handled using a set of strategies to minimise risks of switching containers in a production environment. Blue-Green Deployments emphasize seamless transitions with load balancing two environments, while Rolling Updates prioritize incremental live container updates with minimal service disruption, and Canary Releases focus on gradual exposure and risk reduction by testing new versions with a small user subset [29].

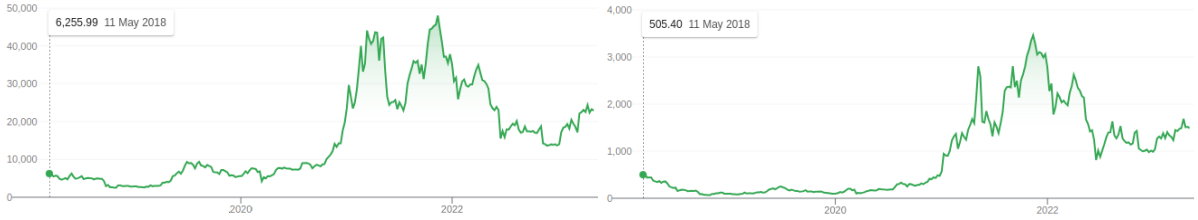
C.6 Containerized Server - Client Connection

There is a multitude of ways a containerized application may communicate with a client and it ultimately depends on the requirements of the system. RESTful APIs allow clients to access resources through standardized URIs and exchange data using JSON or XML formats [70]; even though it may be too high-latency and too low-throughput for real-time or frequent data communication (for example, data streams). Message Brokerage (such as Apache Kafka) is essentially middleware between server and client and allows a high degree of flexibility, asynchronous and fault-tolerant messaging [44]. It does however add some complexity and overhead latency. Finally, gRPC and WebSockets both support bidirectional streaming [61] [75], which allows for low-latency and high-throughput interactions, ideal for performant and permanent connections. In general, gRPC may get the edge in performance, though it is more complex to implement.

Appendix D

Data Methods

Figure D.1: Bitcoin and Ethereum price over 5 years



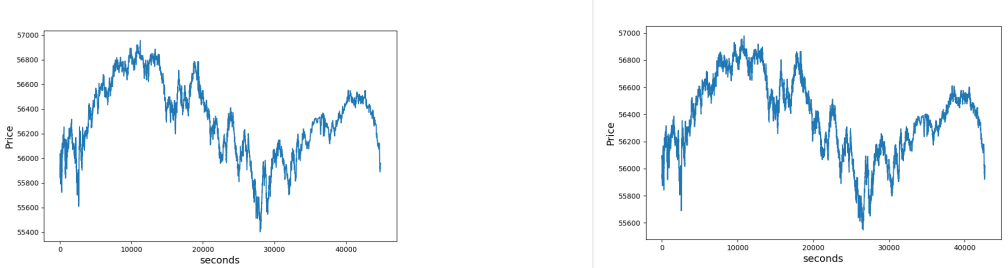
(a) Bitcoin price

(b) Ethereum price

Table D.1: Feature set

feature	description (i = index, n - levels)
$v_1 = \{(P_i^{ask} + P_i^{bid})/2\}_{i=1}^{n}$	midpoint price
$v_2 = \{(P_i^{ask} - P_i^{bid})/2\}_{i=1}^{n}$	price spread
$v_3 = \{(V_i^{ask}\}_{i=1}^{n=1}$	best ask volume
$v_4 = \{\sum_{i=1}^{n}(P_i^{ask})/n\}$	mean ask price (map)
$v_5 = \{\sum_{i=1}^{n}(P_i^{bid})/n\}$	mean bid price (mbp)
$v_6 = \{\sum_{i=1}^{n}(V_i^{ask})/n\}$	mean ask volume (mav)
$v_7 = \{\sum_{i=1}^{n}(V_i^{bid})/n\}$	mean bid volume (mbv)

Figure D.2: Side by side comparison



(a) Kaggle data midpoint feature

(b) Binance data midpoint after feature extraction

Appendix E

Tranformer Methods

E.1 Data Preparation

Various pre-processing techniques were applied to prepare the processed order book data for training. The initial pre-processing step involved converting the UNIX timestamps column to date/time and transforming the date column into more informative time features, such as month, day, weekday, hour, minute, and second. By incorporating these time features into the model, the model can better account for seasonality, trends, and cyclical patterns typically present in time series data. This improves the model's ability to accurately capture patterns and generate more precise predictions. This process has been previously validated [reference for this].

The subsequent step in pre-processing entails adjusting the input data structure and shape based on whether the data is multi-variate or single-variate. For multivariate data, multiple features are used, while for single-variate data, only the target column is chosen. The resulting data frame contains columns in the form of $['date', ...(otherfeatures), targetfeature]$ for multivariate data and $['date', targetfeature]$ for single-variate data.

Lastly, normalization ensures all feature values are on a similar scale. This is important as the raw data can often have uneven scales, which may cause some features to dominate during training. Normalization also speeds up the training process without compromising performance by guaranteeing that the gradients propagated during backpropagation are more stable and can be calculated more efficiently.

Table E.1: Hyperparameter testing for the transformer models on 3 hour Dataset

Models			Transformer		Informer		Pyraformer	
Metrics			MSE	MAE	MSE	MAE	MSE	MAE
Hyperparameters	Model Dim	256	0.055	0.121	0.064	0.163	0.053	0.119
		512	0.054	0.122	0.064	0.140	0.053	0.118
		1024	**0.048**	**0.119**	**0.061**	**0.135**	**0.050**	**0.109**
		2048	0.051	0.120	0.066	0.116	0.055	0.122
	Attn Heads	2	0.051	0.119	**0.051**	**0.125**	0.053	0.112
		4	0.050	0.121	0.058	0.137	0.052	0.121
		8	0.51	0.117	0.061	0.149	**0.052**	**0.107**
		16	**0.049**	**0.114**	0.067	0.152	0.052	0.111
		32	0.050	0.118	0.074	0.160	0.053	0.116
	Encoder Layers	2	**0.049**	**0.112**	**0.055**	**0.142**	-	-
		4	0.050	0.118	0.066	0.141	-	-
		6	0.052	0.120	0.064	0.149	-	-
		8	0.053	0.129	0.061	0.157	-	-
	Decoder Layers	2	0.053	0.122	0.059	0.134	**0.051**	**0.110**
		4	0.050	0.120	0.057	0.138	0.065	0.117
		6	**0.049**	**0.112**	0.051	0.122	0.068	0.157
		8	0.053	0.120	**0.045**	**0.114**	0.065	0.122
	FFN Dim	512	0.051	0.115	0.063	0.172	0.054	0.117
		1024	**0.049**	**0.112**	0.061	0.129	**0.051**	**0.113**
		2048	0.051	0.117	**0.048**	**0.124**	0.053	0.121
		4096	0.053	0.125	0.054	0.113	0.054	0.114

E.2 Informer

Table E.2: Optimal sampling factor of ProbSparse attention in the Informer

Sampling Factor	MSE	MAE
1	0.059	0.138
3	0.054	0.131
5	**0.051**	**0.127**
8	0.053	0.126
10	0.056	0.129

E.3 Pyraformer

Table E.3: Optimal Number of Coarser Scale Nodes for CCSM module in Pyraformer

Coarser Scale Nodes	MSE	MAE
3	**0.049**	**0.107**
4	0.052	0.114
5	0.052	0.116
6	0.053	0.112
7	0.052	0.121
8	0.05	0.119

Table E.4: Optimal Number of Intra-scale and inter-scale connections in PAM Modle of Pyarformer

Inter-scale & intra-scale	MSE	MAE
[2,2]	**0.048**	**0.118**
[4,4]	0.051	0.114
[6,6]	0.052	0.124
[8,8]	0.061	0.133

Appendix F

Deep Reinforcement Learning Analysis and Results

F.1 Research Findings

Reward Function	A2C			PPO		
	BTC	ETH	LTC	BTC	ETH	LTC
Positional PnL	0.0934	-0.0851	-0.1836	0.1094	0.1320	0.3462
Trade Completion	3.3063	0.0643	1.5078	0.4051	0.0650	1.4223

Table F.1: Comparison of reward functions, different coins and approaches

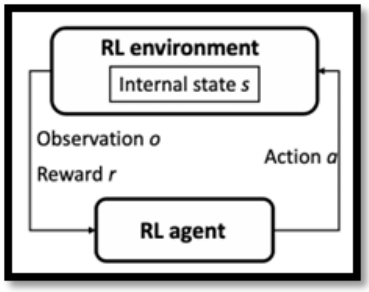


Figure F.1: Reinforcement Learning Basic Components

Evaluation Environment	Baseline	With Tech Indicator	With Sentiments
Accumulated Return	33960 ± 4473	89782 +- 18980	115591 +- 17721
Sharpe Ratio	1.43 +- 0.13	2.75 +- 0.43	3.14 +- 0.4
Commission	355 +- 83	1109 +- 248	1447 +- 268
Sharpe Ratio Benchmark	0.85	1.4	2.4

Table F.2: The Performance Evaluation comparison between three different evaluations and benchmark

F.2 Design and Methods

Network	ActorNet	Value-Critic Net	Actor-Critic Net
Convolutional Layers	2	2	2
Batch Normalization Layers	2	2	2
Max Pooling Layers	1	1	1
Dropout Layers	1	1	1
LSTM layers	1	1	1
Fully Connected Layers	2	2	2
Total Number of Layers	9	9	9

Table F.3: Number of Layers per Network of A2C algorithm

Layers	Profit of Experiment	Actor Loss (using PPO)
Without LSTM layers	-1128.33$	0.304
Model size reduced	-1400.54$	2.686
Learning Rates at 0.001	-1176.75$	5.012
Learning Rates at 0.005	-1136.35$	8.075
RSMprop on actor and SGD on remaining	-1034.52$	1.562
Finalised Model	-905.67$	1.367

Table F.4: Profit after running model through the test set with 100000$ starting funds

Appendix G

Transformer Results

G.1 Tranformer Price prediction Results

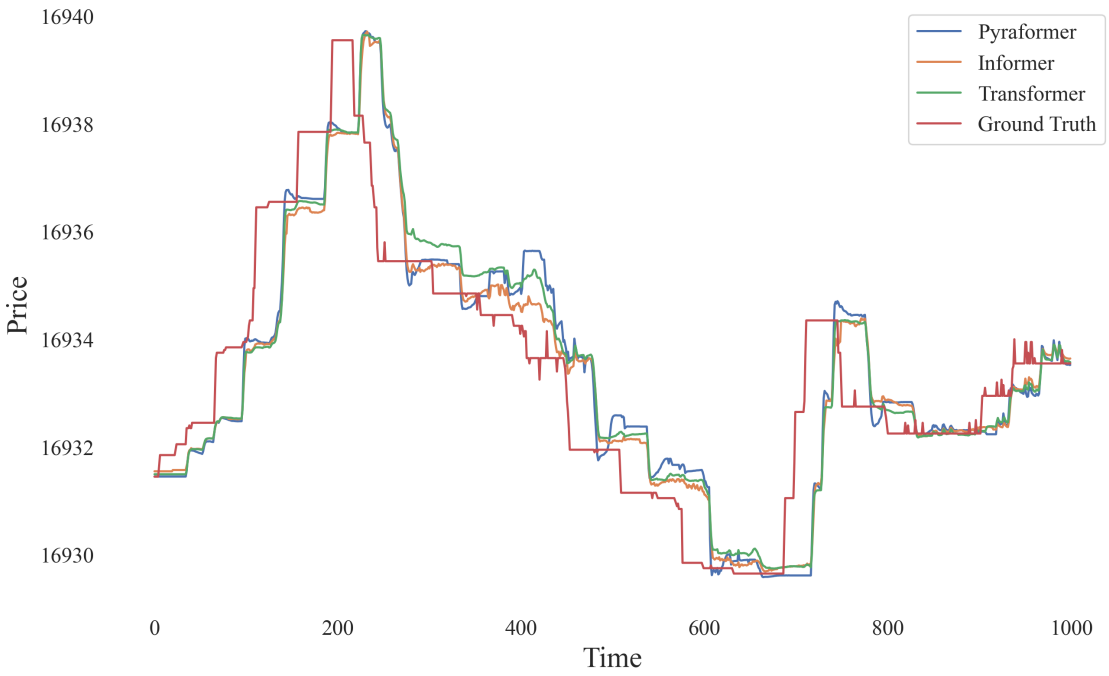


Figure G.1: Prediction Length = 30

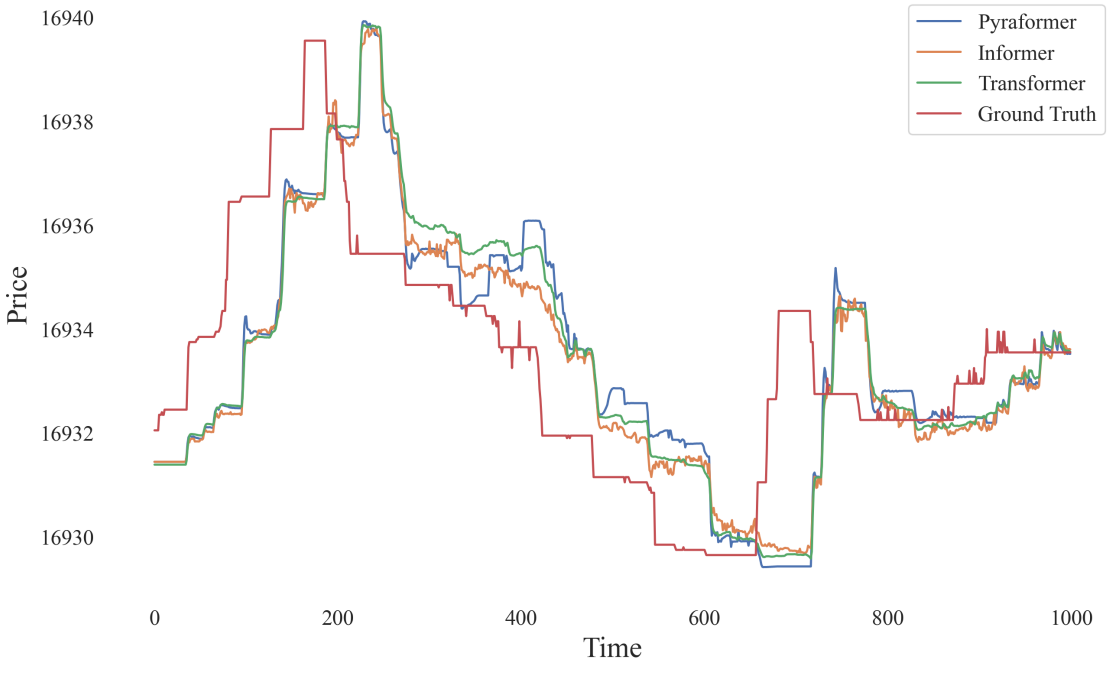


Figure G.2: Prediction Length = 60

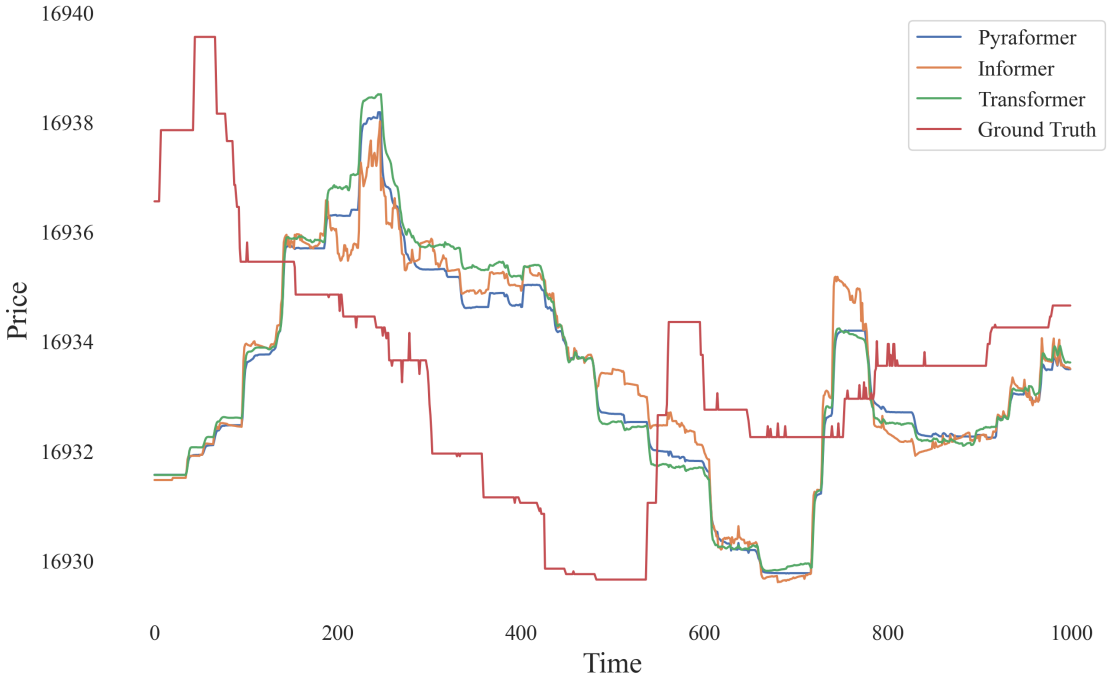


Figure G.3: Prediction Length = 180

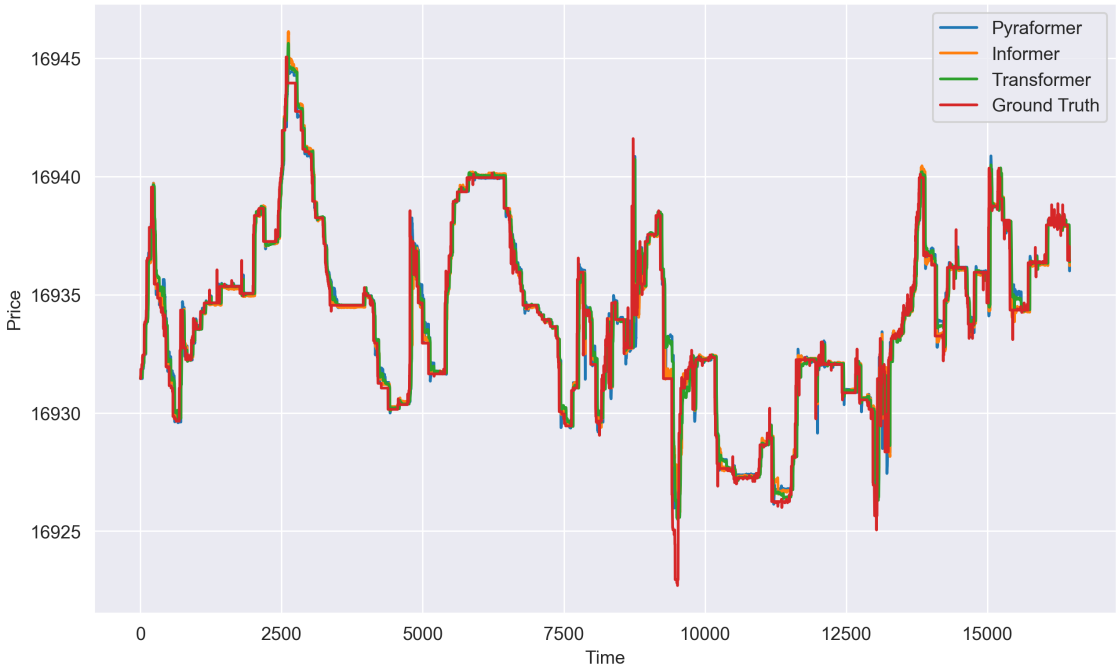


Figure G.4: Prediction Length = 30 , Full test set

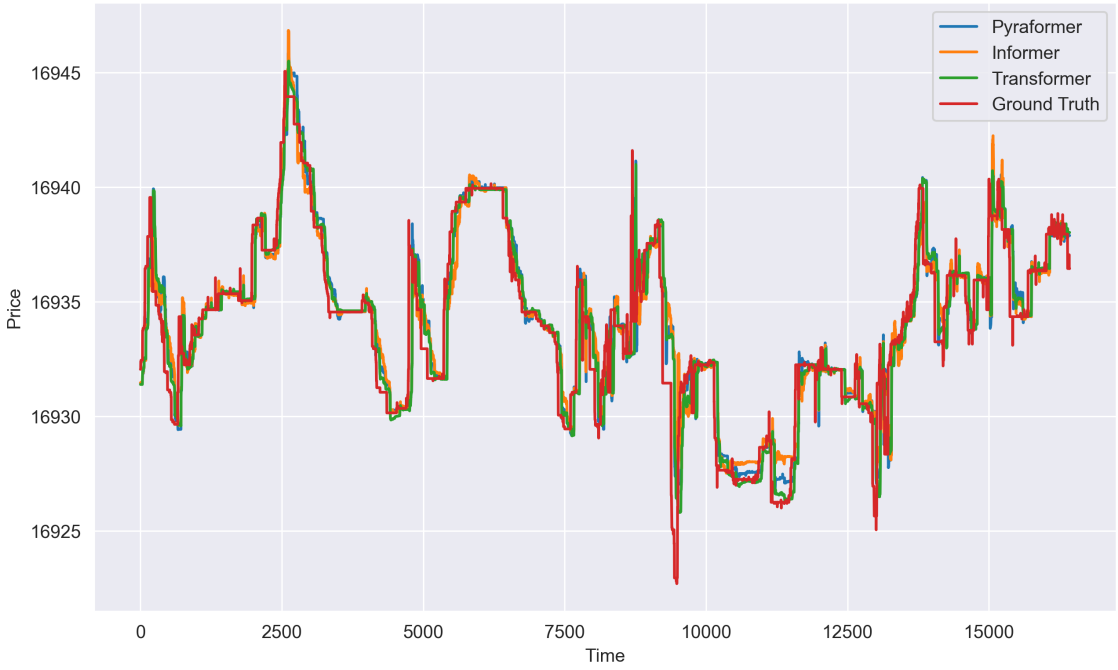


Figure G.5: Prediction Length = 60 , Full test set

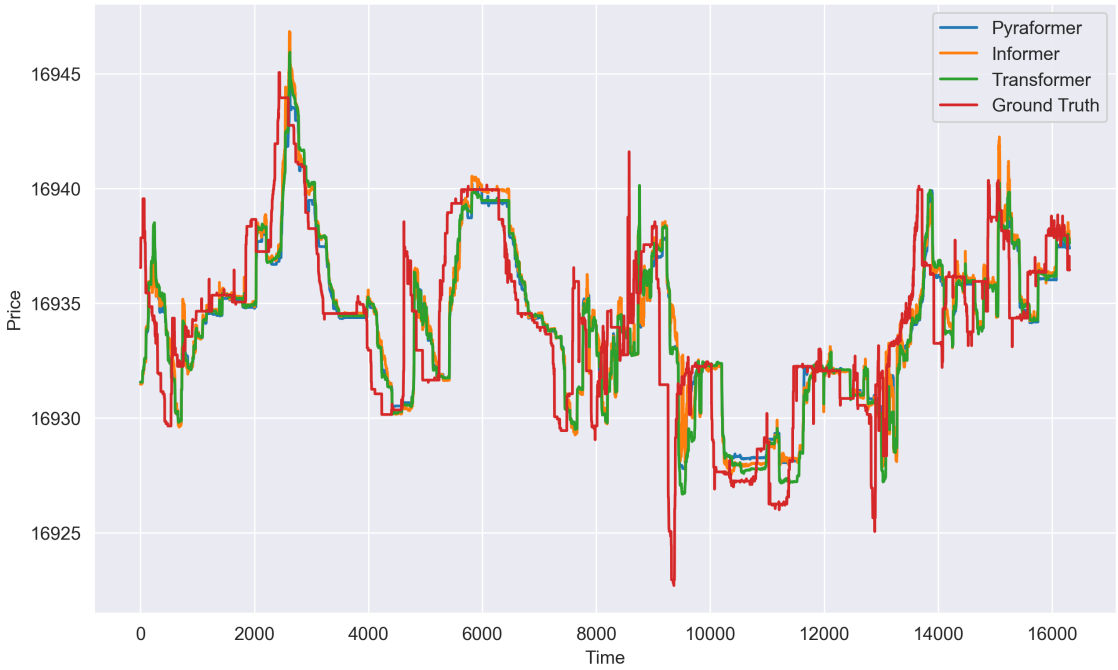


Figure G.6: Prediction Length = 180 , Full test set

G.2 Trading Results

Table G.1: Trading Strategy Performance Dataset 2 (P:Pyraformer, T:Transformer, I:Informer)

	Heuristics			DRL		
Metric	**P**	**T**	**I**	**P**	**T**	**I**
Net Return (%)	-0.000013	-0.000013	-0.000013	-0.000051	-0.000080	-0.000087
Sharpe Ratio	0.000167	-0.000167	0.000167	-0.000586	-0.000632	-0.000724
Max. Drawdown (%)	-0.000017	-0.000017	-0.000017	-0.000053	-0.000080	-0.000089
Max. DD Duration (s)	147s	111s	128s	147s	352s	219s
Exec. Time (s)	364s	364s	364s	364s	364s	364s
Avg. Resp. Time (s)	2.008s	21.195s	7.005s	2.039s	20.391s	7.047s
Num. of Trades	2	2	2	3	6	5

Table G.2: Trading Strategy Performance Dataset 3 (P:Pyraformer, T:Transformer, I:Informer)

	Heuristics 1			DRL		
Metric	**P**	**T**	**I**	**P**	**T**	**I**
Net Return (%)	-0.000152	-0.000206	-0.000152	-0.000084	-0.000114	-0.000081
Sharpe Ratio	-0.001145	-0.001297	-0.001145	-0.000836	-0.000913	-0.000777
Max. Drawdown (%)	-0.000152	-0.000206	-0.000152	-0.000084	-0.000114	-0.000081
Max. DD Duration (s)	237s	232s	220s	357s	291s	336s
Exec. Time (s)	364s	364s	364s	364s	364s	364s
Avg. Resp. Time (s)	2.010s	20.314s	6.794s	2.018s	24.807s	8.040s
Num. of Trades	6	8	6	4	2	1

Appendix H

Client Application

The **design process** for the front-end web app began with the team looking at the design of several trading markets such as Binance, CoinMarketCap, eToro, Coinbase, and Bitfinex. Common features were identified that should be incorporated to the application as well as keeping in mind the theme and layout. From there wireframes and mock-ups (see Appendix H) were drawn using Figma, a collaborative online interface design tool, these mock-ups provided a base and were iterated upon through prototypes. This can be seen from H.3 to H.4, key features that were identified early on, were later scrapped and the design of the front end underwent major changes. The requirements for the design were outlined based on what information was to be relayed to the front end as well as the needs of the end-users.

For the **development process**, the team made the decision to use JavaScript as the primary programming language and React as the front-end framework. This decision was based on the popularity of JavaScript for creating dynamic and interactive web applications, as well as React's extensive library of features that facilitate user interface design through component modularity and state management. By combining JavaScript and React, the team was able to harness the power of JavaScript to build a feature-rich and interactive web application while also taking advantage of React's scalability and reusability for designing user interfaces. To maintain consistency in design, MaterialUI from Google was used as a component library. The team also conducted research on chart libraries and ultimately decided on Lightweight-charts due to its specific purpose for displaying financial data and development by TradingView.

The **user interface** design for the front-end web application was created with simplicity and modularity in mind. The team opted for a dark theme to reduce eye strain and improve the visibility of the financial data displayed on the application. The layout of the user interface was compartmentalized, allowing for easy customization and addition of new features. The design was intended to be intuitive and user-friendly, with minimal distractions and a focus on the key features of the application.

The navigation and **functionality** of the front-end web application were designed to be intuitive and efficient. The application works based on the Server-Client Custom JSON API (see Appendix I), through this continuous connection to the back end server, a live stream of financial data and portfolio information is provided to the front end and appropriately displayed. Through the use of interactive elements such as selectors and buttons the end user is able to configure the underlying trading strategies and datasets utilised. There are Interactive graphs with markers that provide users with a visual representation of the trading strategies and it's decisions. Additionally to provide the end user with some information regarding the back end, there is a short description on key areas found in the project section.

To see the client in action there is a video demonstration that can be found at: Link to video demo otherwise the gitlab repository provides the necessary source files to run the client locally.

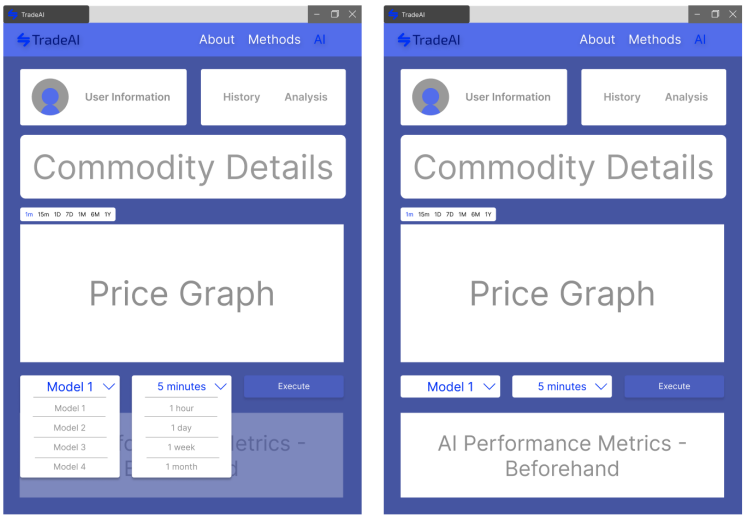


Figure H.1: Wireframe 1

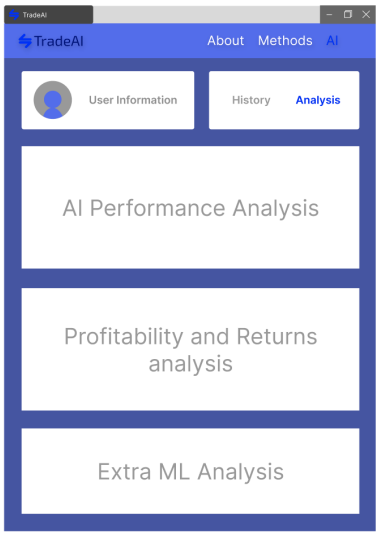


Figure H.2: Wireframe 2

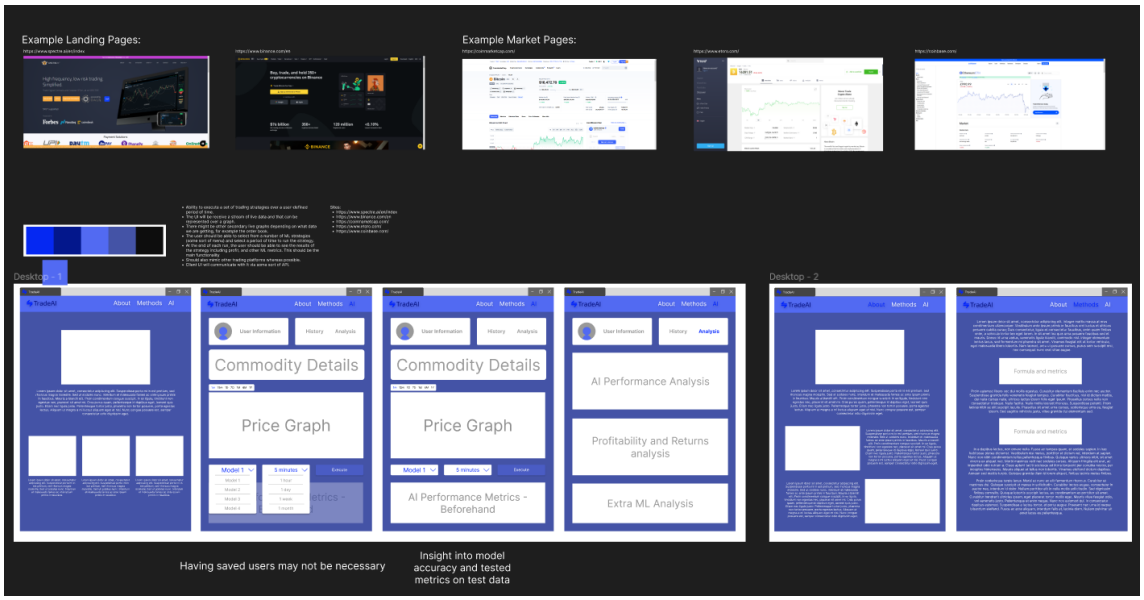


Figure H.3: Figma board 1

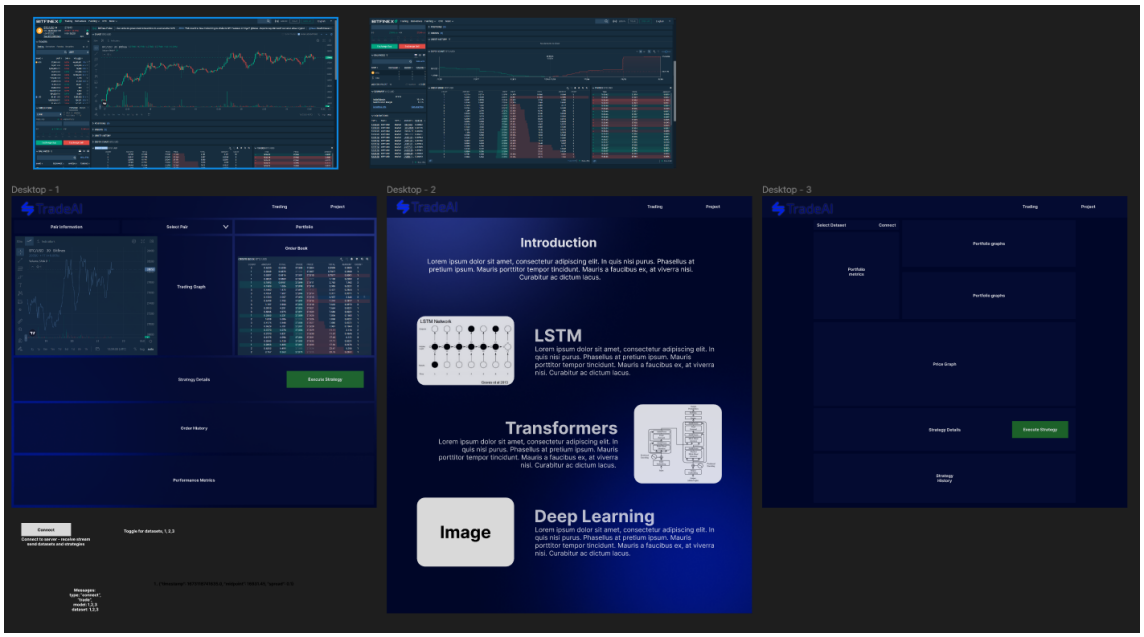


Figure H.4: Figma board 2

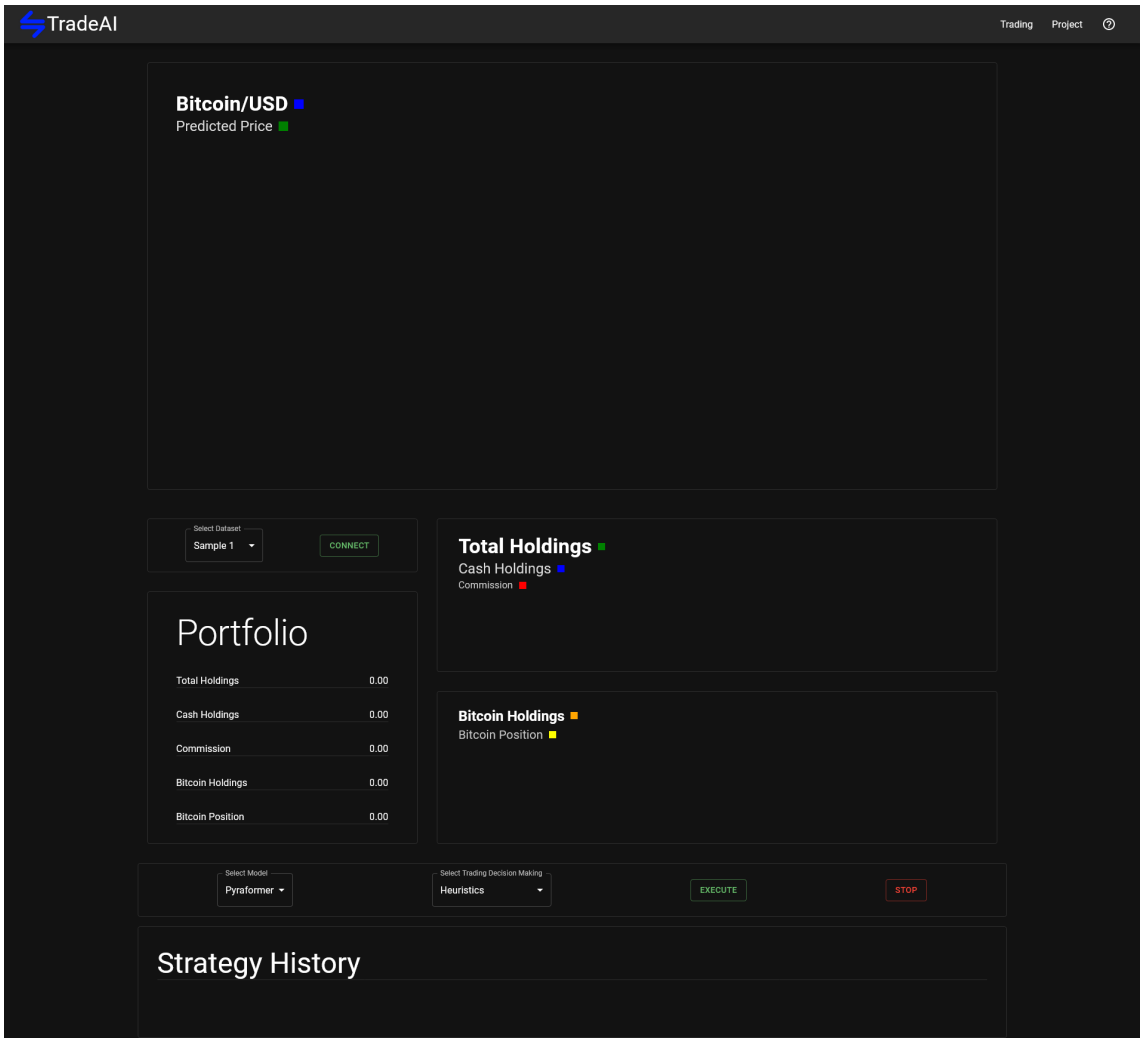


Figure H.5: Empty client web application

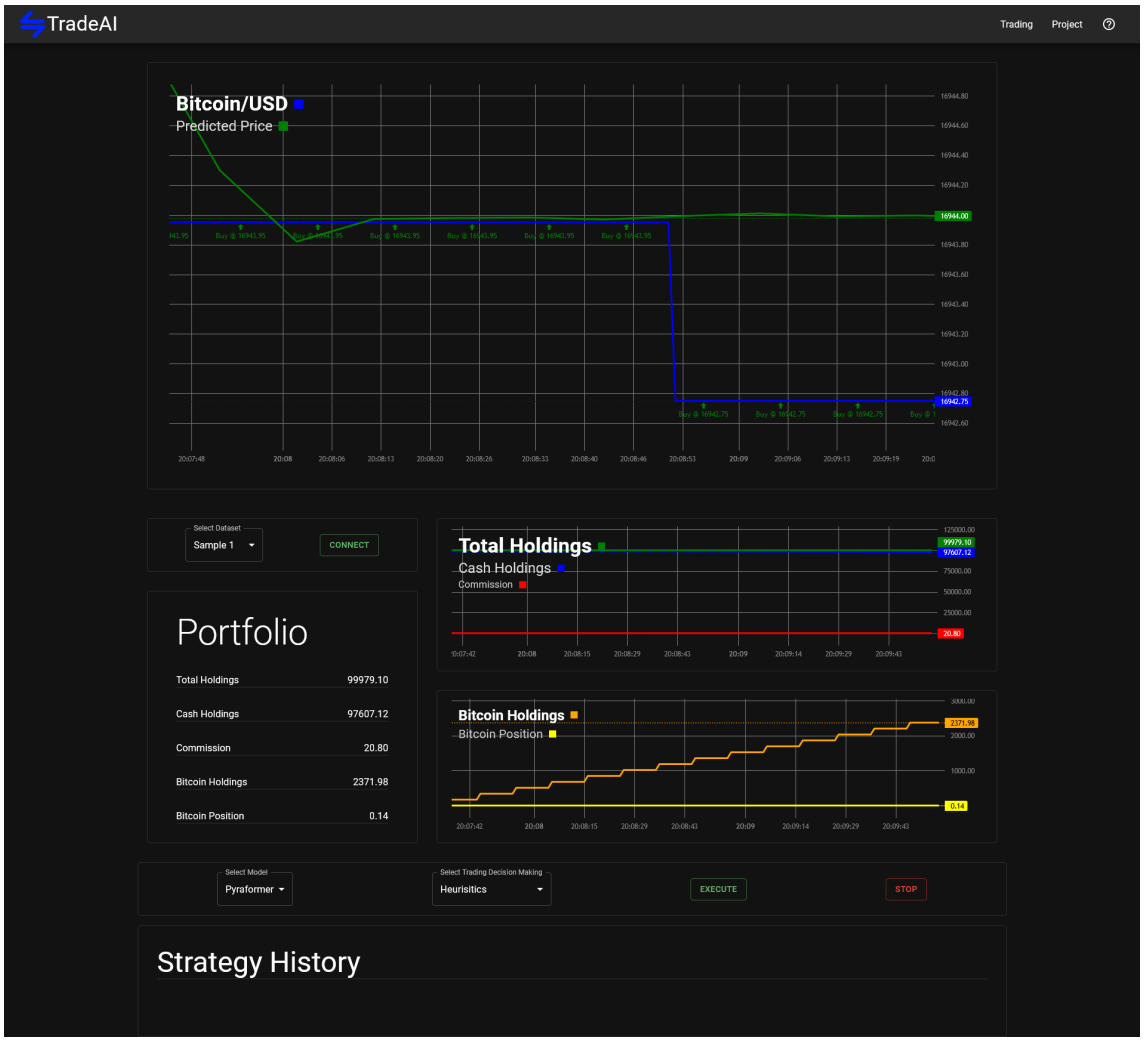


Figure H.6: Populated client web application

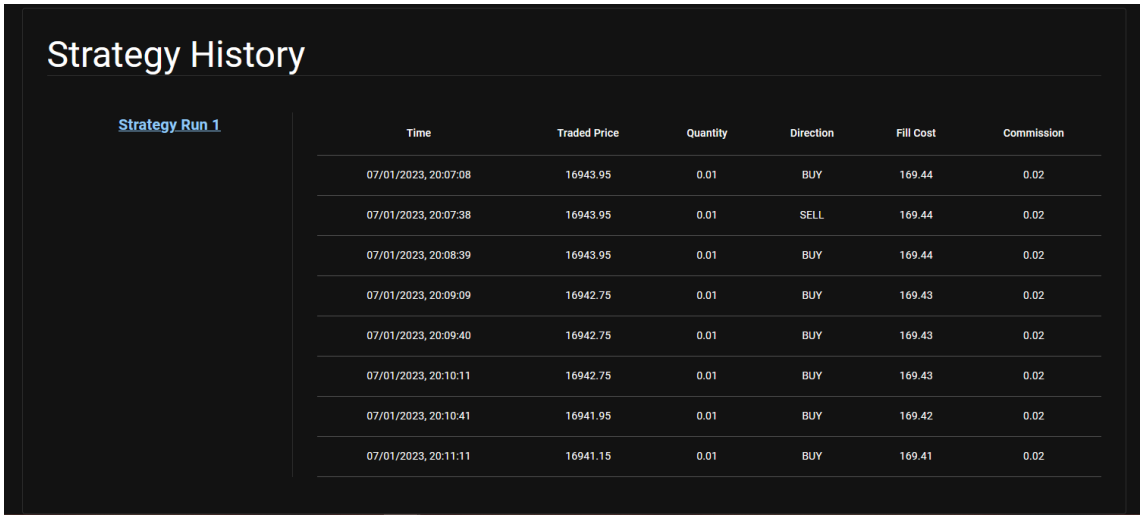


Figure H.7: Populated Strategy History

Appendix I

Server Client Custom JSON API

Message Type	Payload	Purpose

CONNECT	{
	"type": "connect",
	"dataset": ["sample-1", "sample-2", "sample-3"]
	}
		Initalises the connection with the server and specifies the dataset to be used.

START TRADE	{
	"type": "trade",
	"alpha": ["pyraformer", "informer", "transformer"],
	"trading_decision_making": ["heurisitics", "drl"]
	}
		Informs the server to start a trade with the specified trading strategy and decision making algorithms
		alpha: Refers to the available machine learning models providing the predicted price.
		decision_making: The underlying trading strategy that sends trade signals.

STOP TRADING	{
	"type": "stop_trading"
	}
		Informs the server to stop trading.

Table I.1: Server JSON API

Message Type	Payload	Purpose

LOG CONNECTION	{
	"type": "log",
	"message": "Connected to server"
	}
		Confirms that the client has successfully connected to the server

MARKET AND PORTFOLIO	{
	"type": "market_and_portofolio",
	"timestamp": 1673122014380,
	"midpoint": 16943.95,
	"spread": 0.1,
	"btc_holdings": 0,
	"cash_holdings": 100000,
	"commission_holdings": 0,
	"total_holdings": 100000,
	"btc_position": 0
	
	}
		Provides market and portfolio information to the client
		timestamp: Time relating to price as Unix Timestamp.
		midpoint: Price calculated as midpoint from orderbook.
		spread: Difference between immediate buy price and immediate sell price.
		btc_holdings: Current portfolio bitcoin holding value.
		
		cash_holdings: Portfolio spending power.
		commission_holdings: Total captial spent on commissions during trades.
		total_holdings: Sums cash holdings and bitcoin holdings to provide total capital held
		btc_position: Amount of bitcoin held

PRICE PREDICTION	{
	"type": "price_prediction",
	"timestamp": 1673122130393
	"timestamp_prediction": 1673122160393
	"price_prediction": 16943.978515625,
	"stop": 0,
	"isDrl": 0
	
	}
		Sends a price prediction to the client along with metadata such as
		timestamps and whether the prediction was generated using
		Deep Reinforcement Learning (DRL) or not.
		
		timestamp: Time of generating price prediction
		timestamp_prediction: Time of the predicted price
		price_prediction: Predicted price generated
		stop: Server signal to indicate stopping of trading
		isDrl: Indicates whether DRL was used.

FILL DATA	{
	"type": "fill",
	"timestamp": 1673122142452,
	"traded_price": 16942.75,
	"symbol": "BTC",
	"quantity": 0.001,
	"direction": "BUY",
	"order_type": "MKT",
	"fill_cost": 169.4275,
	"commission": 1.3
	}
		Sends fill data to the client for a trade that was executed.
		
		timestamp: Time of trade execution.
		traded_price: Price traded at.
		symbol: Cryptocurrency being traded.
		quantity: Amount of currency exchanged.
		direction: Indicates Buying or Selling.
		order_type: Refers to the type of order, market or limit order.
		fill_cost: Cost of trade.
		commission: Fees from exchange