Applying Recurrent Neural Network to predict forex prices for retail traders: High-Frequency directional trading approach

Masixole Boya

Supervisor(s):
Dr. Martins Arasomwan

A research report submitted in partial fulfillment of the requirements for the degree of MSc Data Science

in the

School of Computer Science and Applied Mathematics
University of the Witwatersrand, Johannesburg

13 November 2024

Abstract

This research investigates the application of Recurrent Neural Networks (RNNs), specifically Long Short-Term Memory (LSTM) architectures, for predicting forex prices in a high-frequency trading environment tailored for retail traders. The study addresses the significant gap in existing literature regarding the utilization of high-frequency trading strategies by individual traders, who often face challenges in accessing sophisticated predictive tools. Through the development and evaluation of two LSTM-based models, this research aims to enhance the understanding of how machine learning techniques can democratize trading strategies in the forex market. Model 1, optimized through a random search technique, demonstrated superior performance compared to Model 2, which was based on established literature and empirical adjustments. Key performance metrics indicated that Model 1 achieved lower test loss, Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE), despite both models exhibiting poor R-squared values. These findings underscore the potential of well-optimized LSTM models to effectively capture short-term price movements within a one-minute timeframe, providing retail traders with actionable insights for informed decision-making. This study contributes to the growing body of knowledge on machine learning applications in finance and highlights the importance of equipping retail traders with advanced tools to navigate the complexities of the forex market. Future work is recommended to explore hybrid modeling approaches and real-time implementation strategies, further enhancing predictive accuracy and applicability across various asset classes.

Declaration

I, Masixole Boya, declare that this proposal is my own, unaided work. It is being submitted for the degree of MSc Data Science at the University of the Witwatersrand, Johannesburg. It has not been submitted for any degree or examination at any other university.

Masixole Boya

13 November 2024

Acknowledgements

I appreciate Dr. Martins Arasomwan's understanding and patience, guidance and support, which will help make this research project a learning experience.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Asset classes are fundamental categories of financial instruments that share similar characteristics and regulatory frameworks. The primary asset classes include equities (stocks), fixed income (bonds), and cash equivalents (money market instruments). Additionally, some classifications extend to 'alternative assets', which encompass currencies, commodities, and real estate [22]. Understanding these asset classes is crucial for effective investment strategies and portfolio management, as they each exhibit distinct risk-return profiles and market behaviors [27].

This study focuses on the asset class of currencies, which are recognized as accepted mediums of exchange within their respective nations. Prominent examples include the US Dollar (USD), British Pound (GBP), and Euro (EUR), which are among the most traded currencies globally. The rise of digital currencies, such as Bitcoin and Dogecoin, has introduced additional volatility into the currency exchange landscape. Currencies are traded on the Foreign Exchange Market (FOREX), where they are exchanged in pairs based on their relative values [22]. The rates of these pairs fluctuate rapidly due to a combination of factors including economic indicators, geopolitical events, and market sentiment. Currency pairs are typically categorized into "major pairs" (e.g., EUR/USD, GBP/USD) and "minor pairs" (e.g., EUR/GBP, GBP/JPY), depending on their trading volumes against the USD [27].

The mathematical modeling of trading strategies has evolved significantly over time. Initially dominated by traditional statistical methods, the field has seen a paradigm shift with the introduction of Machine Learning (ML) techniques [14].

These advanced algorithms have proven to be more effective in forecasting market trends and making trading decisions. Specifically, ML addresses the challenges traders face in processing vast amounts of market data to determine optimal trading actions [3].

A key focus of this research is High-Frequency Trading (HFT), characterized by rapid execution of trades facilitated by sophisticated algorithms that operate within milliseconds. HFT represents a specialized subset of algorithmic trading that leverages technological advancements to automate trade executions based on real-time data analysis [20]. As highlighted by [2], HFT merges concepts from computer science with high-frequency finance, allowing firms to capitalize on fleeting market opportunities that would be impossible for human traders to exploit effectively [9]. To achieve sustained profitability in this environment, traders must employ advanced strategies rooted in various modeling techniques and machine learning methods, including both supervised and unsupervised learning approaches [24].

Recent literature emphasizes innovative methodologies for enhancing trading strategies within HFT contexts. For instance, the directional change approach focuses on significant price movements rather than fixed time intervals, enabling traders to capture deeper insights into market dynamics and movements [17]. This is to underscore the potential for ML models to enhance decision-making processes in HFT settings.

1.2 Problem Statement

The primary problem addressed in this research is the lack of comprehensive studies on HFT as it pertains to retail traders — individuals who trade using their own capital rather than through large financial institutions or trading firms. Most existing literature focuses on the practices and technologies employed by institutional traders, leaving a significant gap in understanding how retail traders can effectively utilize HFT strategies. This oversight is critical, as retail traders face unique challenges, including limited access to sophisticated trading technologies and market data compared to their institutional counterparts. As such, there is a pressing need for research that specifically explores how retail traders can leverage HFT techniques to enhance their trading performance and capitalize on market opportunities.

1.3 Research Questions

How can Recurrent Neural Networks be applied to predict forex prices effectively for retail traders to predict prices in a high-frequency trading environment (within one minute)?

1.4 Research Aims and Objectives

1.4.1 Research Aims

The aim is to develop a predictive model using Recurrent Neural Networks (RNNs) to forecast forex prices for retail traders, for a high-frequency directional trading strategy.

1.4.2 Research Objectives

The objective is to design and implement a Recurrent Neural Network model that accurately predicts short-term one-minute forex price movements.

1.5 Significance of the study

The significance of this study lies in its potential to bridge the knowledge gap regarding HFT strategies for retail traders. By focusing on the application of Recurrent Neural Networks (RNNs) within a high-frequency directional trading framework, this research serves as a step forward in the direction of understanding how the implementation of machine learning techniques in trading can democratize access to sophisticated trading strategies, enabling individual traders to compete more effectively in the forex market. As the forex market continues to grow and evolve, equipping retail traders with effective tools and strategies becomes increasingly important for fostering a more inclusive trading environment.

1.6 Limitations

The limitation is mostly the unavailability of high-frequency data at a level less than one minute from currently accessible APIs. Whether it is for trading markets like FOREX or cryptocurrencies, free data sources typically offer samples at a minimum frequency of one minute. Thus, even though the aim is to tap into the domain of trading firms, which trade in order of milliseconds, the lowest level we could look at is at the minute level.

1.7 Definitions

Currency Pair: A currency pair refers to the exchange rate relationship between two currencies, where one currency's value is expressed in terms of the other. For instance, EUR/USD represents the exchange rate between the Euro and the US dollar.

Base Currency: The base currency typically denotes the currency of the trader's home market in which buying or selling occurs.

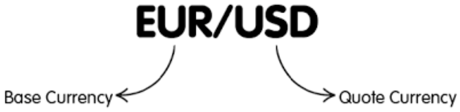


FIGURE 1.1: Base currency and quote currency

Long Position: A long position is the buying of a pair, which will only be yielding a profit when the price of the pair goes up.

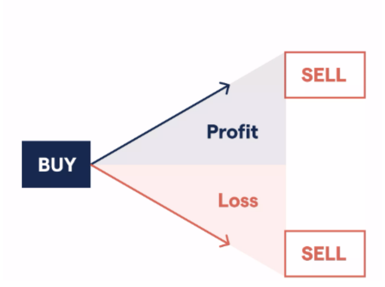


FIGURE 1.2: Summary of a 'long' position, where you **buy**, and if the price goes up, you make a profit when you sell later. You make a loss if the price goes down and sell later.

Short Position: A short position is the selling of a pair, which will only result in a profit when the price goes down.

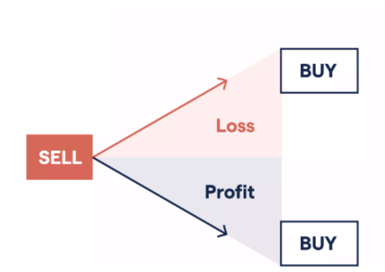


FIGURE 1.3: Summary of a 'short' position, where you **sell**, and if the price goes down, you make a profit when you sell later. You make a loss if the price goes up and sell later.

Stop Loss: A stop loss order specifies the price at which a trader closes their position, either a long or short position, to exit the market and minimize losses in case the trade moves against their expectations. It serves as a crucial tool for risk management.

1.8 Overview

This report presents a comprehensive investigation into the application of Recurrent Neural Networks (RNNs), specifically Long Short-Term Memory (LSTM) architectures, for predicting forex prices in a high-frequency trading environment aimed at retail traders. The research addresses the significant gap in existing literature regarding the utilization of high-frequency trading strategies by individual traders, who often lack access to sophisticated predictive tools. The study aims to develop an effective predictive model that accurately forecasts short-term price movements within a one-minute timeframe, thereby equipping retail traders with actionable insights for informed decision-making. The methodology involved training and evaluating two distinct LSTM models, with Model 1 optimized through a random search technique and Model 2 based on established literature and empirical adjustments. Key performance metrics such as test loss, Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE) were employed to assess the models' effectiveness. The results indicated that Model 1 significantly outperformed Model 2, highlighting the importance of model architecture and hyperparameter tuning in achieving accurate predictions. The significance of this study lies in its potential to democratize access to advanced trading strategies for retail traders, fostering a more inclusive trading environment in the evolving forex market. By leveraging machine learning techniques, this research contributes to the growing body of knowledge on financial modeling and underscores the importance of equipping individual traders with effective tools to navigate the complexities of high-frequency trading. Future work is recommended to explore hybrid modeling approaches and real-time implementation strategies, further enhancing predictive accuracy across various asset classes.

Chapter 2

Literature Review

2.1 General context

In the world of high-frequency algorithmic trading, which, as have been characterized, is where financial markets move at breakneck speed, the pursuit of lucrative trading strategies is the business of the day. HFT albeit not a new term, has seen so much growth in the recent decade, such that automated algorithms easily have the capacity to execute trades within the order of milliseconds [5]. Interestingly, early HFT advancements were not always this way.

HFT emerged as stock exchanges transitioned to computerized systems and gained momentum with the introduction of Electronic Communication Networks (ECNs) and the Small Order Execution System (SOES) [21]. ECNs revolutionized trading by speeding up transactions and reducing costs. However, they also introduced challenges. For instance, if an ECN displayed a different price than a regulated market, trades might not execute at the best available price, creating arbitrage opportunities. Early HFT strategies capitalized on these inefficiencies, with traders exploiting information advantages to profit from market disparities. The 1990s saw the rise of SOES Bandits, traders who executed numerous rapid trades to exploit small price fluctuations and delays in market makers' updates. Regulations were introduced to enhance market fairness and transparency, setting the stage for the expansion of HFT [21]. Before the 2008 financial crisis, HFT witnessed exponential growth, with substantial increases in trading volumes, particularly in foreign exchange and interest rate futures. However, post-crisis, growth slowed as escalating infrastructure costs, intensified competition, and stricter regulations dampened enthusiasm for HFT [21].

Trading strategies are a necessity and they are of different complexities and they focus on analyzing different parameters in the trading markets [1].

2.2 Current state in literature

The majority of research on HFT algorithmic trading has been done around the legalities and regulation, over-fitting, strategy development and testing, best indicators, and model loss reduction [19, 23, 25]. What necessitates regulation on strategy development is that HFT is mostly carried out in two ways - harmful (to the market) strategies, and beneficial (again, to the market). Strategies that are beneficial for market quality almost always count in arbitrage, and trend-following strategies [19, 28]. Statistical arbitrage, aims to exploit short-term price divergences between similar instruments, identifying temporary price disparities and executing trades quickly to capture profit opportunities, leveraging speed and real-time data analysis. Directional trading strategies identify short-term trends or momentum in asset prices and make predictions based on anticipated price changes. Market-making provides liquidity to the market by placing limit orders on both sides of the market price. HFT market-makers utilize automated liquidity provision strategies, rapidly placing, canceling, and replacing bid and ask limit orders to profit from resulting spreads [15, 28].

On the other hand, [28] goes on to describe the following harmful strategies that exploit others unfairly through tactics like front-running, spoofing, layering, and quote-stuffing. Front-running involves trading ahead of large orders to profit from anticipated price changes, driving up transaction costs for others. Spoofing and layering manipulate the market by posting fake orders to mislead traders, creating artificial demand or supply. Quote-stuffing floods the market with orders, slowing down competitors and disrupting trading. These strategies harm market integrity and hinder fair trading practices [28].

Of course, of the above-mentioned types of strategies, we want to only practice the beneficial for the market. An actual trading strategy that will decide when to place a trade will still need to either be developed or adopted. [13] provide an in-depth examination of algorithmic trading strategies, and in-turn advocate for a structured approach where the most important thing to take into account when developing strategies is deciding the entry and exit rules, risk management technique, and position sizing. Concurrently, [25] shed light on the pivotal role of indicators in strategy development, especially for the utilization of machine learning algorithms and they present financial indicators derived from minute-level price data for Bitcoin trading. Among these indicators, are Williams $\%R$, and On-Balance Volume (OBV), to inform trading decisions and enhance profitability. Together, these insights highlight the multifaceted nature of algorithmic trading strategies.

[6] undertook a very comprehensive systematic review, on 60 scientific publications from 2010 to 2021, on what the industry norms are when it comes to training ML models for financial market forecasting and why. The analysis of articles reveals a diverse landscape of machine learning methodologies employed in forecasting. Notably, Long Short-Term Memory Neural Network (LSTM) and Artificial Neural Network (ANN) emerged as the most prevalent algorithms, each utilized by 19 primary studies. Although [23] found Ensemble learning methods to perform better as they are less susceptible to overfitting as they combine multiple weak learners to make one strong final learner. Other algorithms such as Convolutional Neural Network (CNN), Support Vector Machine, and Gated Recurrent Unit (GRU) were also explored. The most prevalent of major currency pair was EUR/USD, in primary studies. Evaluation techniques predominantly relied on percentage split validation, while metrics such as Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) were commonly used to assess model performance.

Lastly, with all the technical know-how, it is of no use if there is none or poor portfolio optimization and risk management as this is what enables long-term profit tracking and refinement of profitability strategy [18, 22]. Risk management is crucial in trading to avoid significant losses despite substantial profits. Planning trades with stop-loss and take-profit levels is a common strategy among successful traders, limiting losses to a small percentage of total capital, typically around 2%. Leveraging in forex trading can amplify both profits and losses, necessitating careful consideration. Accepting losses, learning from them, and maintaining discipline are key factors for successful trading. Money management, including frequent small stops or gains, varies depending on the trader's personality and is essential for profitability in forex trading. Optimization models tailored for currency trading, include the Markowitz ($M - V$), semi-mean-absolute deviation ($SMAD$), and conditional value-at-risk ($CVaR$) models. Key assumptions include trader risk aversion, operation on a standard Forex trading account, and use of 1:1 leverage, with currency pairs analyzed with USD as the quote currency. Methodology involves defining capital amount invested (C), proportion of capital invested in each security (w_i), and unit price of securities (P_{it}), establishing returns (R_{it}), mean (μ_i), variance (σ_i^2), and covariance (σ_{ij}), and introducing the performance function $F_\alpha(w, \gamma)$. The study utilizes convex programming formulations to effectively address the optimization problem, providing theorems and proofs to elucidate the mathematical underpinnings, showcasing the SMAD portfolio's superior risk reduction and profitability over traditional models [18, 22].

2.3 Gaps and Limitations

The profitability of high-frequency trading depends so much on speed, hence leading trading firms to invest in having the fastest hardware and network infrastructure [16]. However, trading algorithms must sacrifice complexity for speed, creating a trade-off in designing optimal high-frequency trading strategies [16]. This is a limitation truly worth noting, as it implies that the speed of a trading algorithm/strategy is limited by its complexity, and to really maximize its speed (or minimize latency), it would have to minimize its complexity.

Existing scientific literature tends to only investigate HFT in the context of big financial institutions, banks, and trading firms but never in the context of retail traders. A retail trader is an individual investor who engages in trading activities through retail commission-free brokerage platforms [7]. These traders are typically young, tech-savvy, often first-time investors, have smaller account balances and are more likely to have strategies based on speculation [7]. Perhaps [8] come close to investigating retail traders in the context of HFT. Still, they consider high-frequency traders and retail traders to be separate entities and they study the interaction between these entities. They characterize retail traders as "noise traders" whose actions could potentially introduce volatility and inefficiencies into the market. They suggest that the uninformed nature of many retail traders’ decisions may lead to herding behavior or other predictable patterns, which could adversely affect market quality. However, they highlight how high-frequency traders play a role in mitigating these negative impacts as they provide liquidity to retail orders and thus counteract the potential adverse effects of noise trading by retail investors.

Chapter 3

Research Methodology

3.1 Research design

I employ a combination of experimental and statistical design methods in this research is driven by the need for a comprehensive investigation into HFT strategies. Experimental design facilitates the controlled manipulation of variables, such as trading indicators and machine learning algorithms, while statistical design methods enable rigorous data analysis, including exploratory data analysis (EDA), feature engineering, and model evaluation. Integrating these approaches allows for experimentation with various strategies and algorithms and rigorous analysis of their performance, and latency implications.

3.2 Data Collection

The data used in this research consists of freely available forex market data obtained through the MetaTrader 5 (MT5) API. The MT5 platform is highly suitable for developing automated trading applications and algorithmic strategies, making it ideal for research in HFT. In this study, minute-level data is requested. The data spans from 1994 to the current date, resulting in over 9 million data points. The choice of a 1-minute sampling frequency aligns with the focus on minute-by-minute price forecasting, a core focus of this study.

3.2.1 Data Source

MT5 is a comprehensive trading platform that supports a wide range of financial instruments, including forex, commodities, stocks, and cryptocurrencies. It offers extensive analytical tools for market data, enabling users to execute trades, develop custom indicators, and build automated trading systems. The MT5 platform is particularly popular among traders and researchers for its algorithmic trading support, among other features and tools.

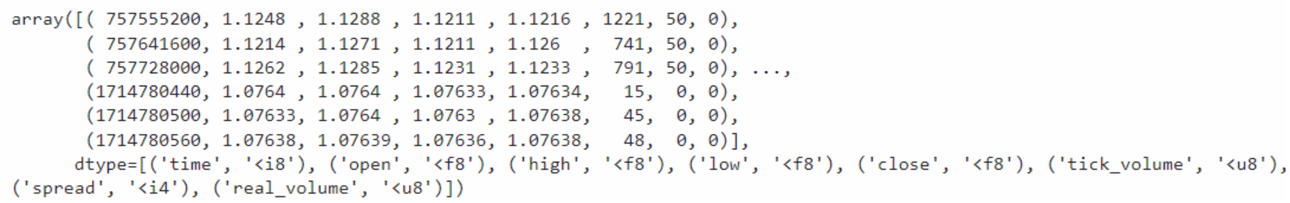
For this research, the MT5 API is leveraged to access historical and real-time forex market data. The Python integration provided by the API is particularly beneficial for this study, as it allows easy data retrieval from the MT5 terminal into a Python environment. This integration supports a wide range of data analysis tasks, from simple statistical calculations to complex machine learning workflows.

The MT5 API provides access to several key data points necessary for algorithmic trading research, including: price data (Bid, ask, and close prices for forex pairs), volume, and timestamp.Retrieving the large historical dataset provided by MT5 (spanning over 30 years) is an attempt to ensure that the model has sufficient data to capture diverse market conditions, ranging from stable periods to highly volatile ones.

For more information, refer to the official documentation at:
MetaTrader 5 Python API Documentation.

3.2.2 Data Description

Return Data



```
array([( 757555200, 1.1248 , 1.1288 , 1.1211 , 1.1216 , 1221, 50, 0),
       ( 757641600, 1.1214 , 1.1271 , 1.1211 , 1.126  ,  741, 50, 0),
       ( 757728000, 1.1262 , 1.1285 , 1.1231 , 1.1233 ,  791, 50, 0), ...,
       (1714780440, 1.0764 , 1.0764 , 1.07633, 1.07634,   15,  0, 0),
       (1714780500, 1.07633, 1.0764 , 1.0763 , 1.07638,   45,  0, 0),
       (1714780560, 1.07638, 1.07639, 1.07636, 1.07638,   48,  0, 0)],
      dtype=[('time', '<i8'), ('open', '<f8'), ('high', '<f8'), ('low', '<f8'), ('close', '<f8'), ('tick_volume', '<u8'),
('spread', '<i4'), ('real_volume', '<u8')])
```

FIGURE 3.1: Raw structure of the array returned by the MetaTrader 5 API.

Data Dictionary

TABLE 3.1: Tabular Version of the structured array returned by the MetaTrader 5 API.

Time	Open	High	Low	Close	Tick Volume	Spread	Real Volume
757555200	1.1248	1.1288	1.1211	1.1216	1221	50	0
757641600	1.1214	1.1271	1.1211	1.126	741	50	0
757728000	1.1262	1.1285	1.1231	1.1233	791	50	0
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
1715050560	1.07686	1.07687	1.0768	1.07681	9	0	0
1715050620	1.07681	1.07681	1.07669	1.07673	21	0	0
1715050680	1.07673	1.07676	1.07672	1.07672	11	0	0

The columns of Table 2.1 represent the following:

- **time**: Represents the timestamp of the data point in Unix time format.
- **open**: Indicates the opening price of the asset for the given time period.
- **high**: Denotes the highest price reached by the asset during the time period.
- **low**: Represents the lowest price reached by the asset during the time period.
- **close**: Represents the closing price of the asset for the given time period.
- **tick_volume**: Indicates the number of ticks or transactions that occurred during the time period.
- **spread**: Denotes the spread, i.e., the difference between the bid and ask prices.
- **real_volume**: Represents the real trading volume for the asset during the time period.

3.3 Pre-processing

3.3.1 Feature Engineering

Technical Indicators

As a very important step in forecasting for trading data, I will calculate various technical indicators that are commonly used in literature on trading strategies [6]. Specifically, the following indicators:

- **Simple Moving Average (SMA)**: The simple moving average is the arithmetic mean of the closing prices over a specified window size. It is used to smooth out short-term price fluctuations and highlight longer-term trends.

  $$\text{SMA} = \frac{\sum_{i=1}^{n} \text{Close}_i}{n}$$

  where:

  - n: Number of periods (window size) over which to calculate the simple moving average.
  - Close_i: Closing price at time i.

- **Exponential Moving Average (EMA)**: The exponential moving average gives more weight to recent prices, making it more responsive to new information. It is calculated using the following recursive formula:

  $$\text{EMA}_t = \alpha \cdot \text{Close}_t + (1 - \alpha) \cdot \text{EMA}_{t-1}$$

  where:

  - α: The smoothing factor, calculated as $\alpha = \frac{2}{n+1}$, where n is the window size.
  - Close_t: Closing price at time t.
  - EMA_{t-1}: EMA at the previous time step $t - 1$.

- **Moving Average Convergence Divergence (MACD)**: As described in the introduction:

  $$\text{MACD} = \text{EMA}_{12} - \text{EMA}_{26}\text{Signal Line} = \text{EMA}_9(\text{MACD})$$

  where:

  - EMA_{12}: 12-minute exponential moving average of the closing prices.
  - EMA_{26}: 26-minute exponential moving average of the closing prices.
  - $\text{EMA}_9(\text{MACD})$: Signal line.

- **William %R**: A momentum oscillator that assesses overbought and oversold levels. It ranges from 0 to -100. Values above -20 are considered overbought, while values below -80 are deemed oversold.

  $$\%R = \frac{\text{Highest High} - \text{Close}}{\text{Highest High} - \text{Lowest Low}} \times -100$$

  where:

  - Highest High: Highest price observed during a given period.
  - Close: Closing price of the asset.
  - Lowest Low: Lowest price observed during a given period.

- **Relative Strength Index (RSI)**: an indicator concerned with how fats wre the prices changing. Typically, RSI values above 70 indicate overbought conditions, while values below 30 indicate oversold conditions.

  $$\text{RS} = \frac{\text{Average of x days' up closes}}{\text{Average of x days' down closes}}$$

  $$\text{RSI} = 100 - \frac{100}{(1 + \text{RS})}$$

  where:

  - RS: Relative Strength, ratio of average of x days' up closes to average of x days' down closes.

- RSI: Relative Strength Index, a momentum oscillator ranging from 0 to 100.

Time Dimensions

In time series forecasting, incorporating temporal features is critical, as market behavior is often influenced by specific times of the day [4]. By adding time-related variables, we aim to capture patterns or dependencies that could be linked to the time when trades occur. The following time dimensions were engineered to better model the minute-level data:

Minute of the Day: The feature *minute of the day* is created to represent the time at which the trade occurred, expressed as an integer ranging from 1 to 1440. This feature helps capture the specific minute in a 24-hour day when trades happen. By encoding the time in this format, we can track intraday patterns and account for variations in market activity that might be influenced by specific times, such as market opening and closing hours or high-volume trading periods.

Hour of the Day: In addition to the minute-level granularity, the *hour of the day* feature is included to capture broader hourly patterns in market behavior. This feature assigns an integer value from 1 to 24 based on the hour during which the trade took place. Incorporating this feature is crucial for detecting hourly trends or cyclical market movements that may impact trading strategies, particularly in forex markets where certain hours are known to have higher volatility or liquidity due to overlapping trading sessions across different time zones.

Log-Returns

Price movements in financial markets are often better modeled using returns rather than absolute price values, as returns are stationary and scale-invariant. To represent the price variations in this study, we calculate the *logarithmic returns (log-returns)* of the closing price. Log-returns are a preferred metric for financial data modeling as they capture relative price changes and allow for easier comparison over time, particularly for data with wide-ranging price levels [4]. This ensures the model learns patterns in terms of the changes, not the true prices themselves.

The log-return is computed as:

$$\text{Log Return} = \log\left(\frac{\text{Close}_t}{\text{Close}_{t-1}}\right)$$

where:

- Close_t: Closing price at time t.
- Close_{t-1}: Closing price at the previous time step $t-1$.

This transformation converts the closing price into a continuous rate of return, which is more suitable for statistical analysis and machine learning models. Furthermore, any missing values introduced by the log-return calculation (due to the shift operation) are removed to ensure a clean dataset for model training.

After calculating log-returns, the original `close` price column is removed, and the `log return` column is renamed to `close` to maintain consistency in the dataset structure. This new `close` column now represents the log-return values, which are the primary input for the machine learning model in predicting future price movements.

3.3.2 Preparing Lagged Data

Preparing Lagged Data

As tends to be the case in time series forecasting, past values of the target variable (current close price) and other features (indicators and the time dimensions) were used. This is known as creating lagged data, where data is shifted backward by a specified number of time steps (lags) to form new features based on previous observations. This method helps capture temporal dependencies that are essential for predictive models in financial markets, particularly for short-term forecasting such as high-frequency trading.

To prepare lagged data, the following steps were carried out:

Lagging the Target Variable: The main goal of lagging is to create features that represent past values of the target variable (in this case, the log-returns or 'close' price), which allows the model to learn from past price movements. The target variable is shifted by a specified number of time steps, denoted as ‘lag steps‘. For each ‘lag step‘, a new column is added to the dataset to store the corresponding lagged values of the ‘close‘ price.

Lagged DataFrame Construction: A new DataFrame, `lagged df`, is created to store the lagged values. The index of this DataFrame is set to be the same as the original DataFrame, ensuring that the time indices are preserved. The following steps detail how the lagged DataFrame is constructed:

- **DateTime:** A new column, `DateTime`, is created to store the timestamp for each observation, copied directly from the index of the original DataFrame. This ensures that the temporal ordering of the data is maintained.
- **ActualValue:** A column named `ActualValue` is added to the lagged DataFrame, which contains the original values of the target variable, the `close` price (or log-returns). This allows for a direct comparison between the actual values and the lagged values during model training.
- **Lagging the Target Variable:** For each specified lag step (from 1 up to the total number of ‘lag steps‘), new columns are created to store the lagged values of the target variable. These columns are labeled as `PrevValue ı`, where ırepresents the number of time steps lagged. For instance, `PrevValue 1` contains the closing prices from the previous minute, `PrevValue 2` contains values from two minutes ago, and so on. These lagged values are essential for capturing the temporal dependencies in the data.

Lagging Additional Features: Beyond the target variable, it is also beneficial to lag other important technical indicators and time-related features, such as the moving averages, RSI, MACD Histogram, and the minute and hour of the day. Lagging these features enables the model to recognize potential patterns and relationships between past indicators and future price movements. For each feature, the following process is applied:

- For each feature in the set of selected columns, a loop iterates from ‘lag steps‘ down to 1, creating new lagged columns for each feature. The lagged feature columns are labeled with the format `feature name-lag-i`, where `i` indicates the number of lag steps applied.

- For example, `SMA-10-lag-1` represents the simple moving average calculated over 10 periods, lagged by 1 minute, and `EMA-10-lag-2` represents the exponential moving average calculated over 10 periods, lagged by 2 minutes.

Removing missing values: Since creating lagged features involves shifting the data, the first few rows will naturally contain missing values due to the lack of preceding data for certain lag steps. To ensure that the dataset remains usable for model training, rows with any missing values are removed from the lagged DataFrame.

3.4 Methods

3.4.1 Procedure and Decisions

Data Cleaning

The first step of data pre-processing involved removing any duplicate rows in the dataset. If duplicate entries were detected, they were eliminated to ensure the data's integrity. Next, missing values were handled by removing rows containing null entries, thus ensuring that all rows had complete information for analysis.

The data's structure was then standardized by converting the time column into a `datetime` format to facilitate time-series analysis. After this, unnecessary columns that were not relevant to the analysis — specifically `tick volume`, `spread`, and `real volume` — were removed. This step helped focus the analysis on the key OHLC (Open, High, Low, Close) data.

Additionally, any rows with zero OHLC values were identified and removed, as these values were deemed invalid for the analysis. With these steps completed, the dataset was now clean and ready for further analysis.

Exploratory Data Analysis (EDA)

For exploratory data analysis, the first step involved gaining an understanding of the dataset's structure by displaying the data's summary and descriptive statistics, which provided insight into its distribution and key features.

Visual analysis was then carried out by generating pairplots to explore relationships between the OHLC values. A candlestick chart was also plotted to show the price movements within a specific time frame, particularly focusing on the last hour of data to capture real-time insights. Histograms and kernel density estimates (KDEs) were used to investigate the distribution of each OHLC value further, while boxplots provided a visual representation of the variability and outliers in the OHLC data.

Lastly, time-based trends were explored by plotting closing prices for each minute of every day in a month, to observe daily price trends. The month for this observation was chosen at random and it was August 2024. The **assumption** is that any patterns, or lack thereof, observed within this month shall represent trends within all months. The data was then grouped by day of the week, and a detailed comparison was performed to observe how closing prices varied over time, both for individual weekdays and across combined days.

Training the Model

Two Long Short-Term Memory (LSTM) models were trained, and their respective structures/architectures and more are described in detail in section 4.3.1.

Data Preparation: The first step in training the LSTM model involves preparing the dataset. The function `prepare_lstm_data` is used to transform the dataset into a format suitable for LSTM training. Specifically, the DateTime column is excluded from the input features, and the target variable, `ActualValue`, is separated for prediction purposes. This ensures that the model only learns from the relevant features, excluding time-related information, which may introduce noise in a neural network model. The dataset is then converted into `numpy` arrays of type `float32` for efficient processing, as LSTM models require numerical data.

The dataset is split into training, validation, and testing subsets using the `split_data` function, with 70% of the data allocated for training and the remaining 30% split equally between validation and testing sets. This split ensures that the model's performance is evaluated on both unseen data (test set) and intermediate progress (validation set) during training.

Weights and Biases Integration: Weights and Biases (W&B) is used for tracking and logging key hyperparameters, metrics, and model artifacts during training. The project configuration is initialized using the `wandb.init` function. Key hyperparameters include a learning rate of 0.001, 12 epochs for training, a batch size of 64, and the `mean_squared_error` as the loss function. The model architecture is specified as an LSTM-based recurrent neural network.

Model Architecture: The LSTM model is designed using the Sequential API in Keras, and the input is a time series of historical data points. The architecture of the models, inlcuding regularization and activation functions, is described in section Section 4.

Model Training: The model is compiled with the `Adam` optimizer, which is a widely used optimization algorithm in deep learning for adjusting the learning rate during training. The metrics tracked during training include the Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE).

The model is trained using the Keras library with Tensoflow, with the training, validation, and test data. The number of epochs is set to 12, and a batch size of 64 is used to ensure that the model processes a manageable subset of data at each iteration, thus speeding up the training process while still benefiting from mini-batch gradient descent. The W&B integration logs key metrics throughout the training process, which allows for real-time monitoring of the model's performance.

Model Evaluation After training, the model is evaluated on the test dataset. The evaluation function returns several metrics, including the loss, MAE, MSE, RMSE. By tracking multiple metrics, the evaluation process provides a comprehensive understanding of the model's performance. Predictions on the test set are computed and compared against the true target values. The Root Mean Squared Error (RMSE) is manually calculated to verify consistency with the Keras output. Furthermore, the R-squared (R^2) value is computed to measure the goodness of fit, which reflects how well the predictions match the actual values.

3.5 Tech Stack and Dependencies

This project utilizes a comprehensive tech stack, focusing on modern deep learning frameworks, data manipulation libraries, and tools for experiment tracking. The following outlines the key components:

- **tmux:** A terminal multiplexer is used to keep the training processes running even after disconnecting from the remote machine. This was especially useful as the training was done on a remote machine, the Wits HPC cluster.
- **Wits HPC Cluster (Linux server):** The project's model was trained on a high-performance computing (HPC) cluster at the University of the Witwatersrand, which required remote access via SSH. The use of `tmux` ensured that long-running processes continued in the background during model training on this cluster.
- **Python:** The core programming language used for all processes in the project.
- **Pandas:** Used for data manipulation and preprocessing, including the management of time-series data and handling of missing values.
- **NumPy:** Used for handling numerical data efficiently such as converting data types.
- **MetaTrader5 (MT5) API:** The MT5 API is utilized for retrieving the long-term historical data.
- **MetaTrader5 Terminal:** The desktop application is required for accessing large historical data via the MT5 API, as it facilitates data downloads directly from brokers. Without the application, the API returns no data.
- **TensorFlow:** The main framework for building and training deep learning models, specifically Long Short-Term Memory (LSTM) networks. It also provides utilities for calculating metrics like mean squared error and directional accuracy.

- **Keras:** A high-level API integrated with TensorFlow, used to define the neural network architecture and manage the training process.
- **WandB (Weights and Biases):** A tool for experiment tracking, logging hyperparameters, and recording metrics during model training. It allows for seamless integration with TensorFlow/Keras and provides real-time monitoring of training performance.
- **Scikit-learn:** Used for splitting the dataset into training, validation, and test sets with `train_test_split`. Additionally, Scikit-learn is employed for performance evaluation metrics such as Root Mean Squared Error (RMSE) and R-squared.
- **Matplotlib & Seaborn (optional for visualization):** These libraries can be used for generating visualizations like pair plots and candlestick charts for Exploratory Data Analysis (EDA), although they are not directly present in the provided code.

3.6 Evaluation Criteria

3.6.1 Evaluation Metrics and Interpretation

To evaluate the performance of the model, several key metrics were utilized, each providing different insights into the accuracy and reliability of the predictions. Below is a detailed explanation of each metric, including how they are calculated, their range of values, and how to interpret them.

Test Loss

The *test loss* represents the error calculated on the test dataset using the chosen loss function during model training. In this project, the Mean Squared Error (MSE) loss function was employed. The test loss essentially quantifies the difference between the predicted values and the actual values in the test set. A lower test loss indicates a model that is better at predicting outcomes, while a higher test loss signifies poorer performance.

- **Range:** The test loss can range from 0 to infinity, with 0 being the ideal value, signifying perfect predictions, and larger values indicating higher discrepancies between predicted and actual values.
- **Interpretation:** A low test loss suggests that the model generalizes well on unseen data, whereas a high test loss indicates overfitting (poor performance on new data) or underfitting (inability to capture the underlying patterns in the data).

Mean Absolute Error (MAE)

The *Mean Absolute Error (MAE)* measures the average magnitude of errors between the predicted and actual values without considering the direction of the errors. It is calculated as the average of the absolute differences between predicted and true values.

- **Formula:**
  $$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$
  where y_i is the actual value and $\hat{y}_i$ is the predicted value.
- **Range:** MAE ranges from 0 to infinity. A value of 0 means perfect accuracy, while higher values indicate larger average errors.
- **Interpretation:** Lower MAE values indicate more accurate predictions. The closer the MAE is to zero, the better the model's predictive capability. Since MAE does not square the errors, it gives a straightforward interpretation of average error in the same unit as the original data, making it easier to understand in a real-world context.

Root Mean Squared Error (RMSE)

The *Root Mean Squared Error (RMSE)* provides a measure of the average magnitude of errors, similar to MAE, but gives greater weight to larger errors. It is computed by taking the square root of the average of the squared differences between predicted and actual values.

- **Formula:**
$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

- **Range:** RMSE ranges from 0 to infinity, with 0 representing a perfect model. Higher values indicate greater error magnitudes.

- **Interpretation:** Since RMSE squares the errors before averaging, it is more sensitive to outliers than MAE. A low RMSE value indicates that the model's predictions are close to the actual values. The advantage of RMSE is that it penalizes large errors more heavily, making it useful when larger errors are more undesirable in the context of the problem.

R-squared (R^2)

The *R-squared* metric, also known as the *coefficient of determination*, measures the proportion of variance in the dependent variable (actual values) that is predictable from the independent variables (features). It provides an indication of how well the model explains the variation in the target data.

- **Formula:**
$$R^2 = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2}$$
where $\hat{y}_i$ is the predicted value, y_i is the actual value, and $\bar{y}$ is the mean of the actual values.

- **Range:** The R^2 value ranges from negative infinity to 1:
  - A value of 1 indicates that the model perfectly explains all the variance in the data.
  - A value of 0 means that the model explains none of the variance, which is equivalent to simply predicting the mean of the target values.
  - Negative values can occur when the model performs worse than a simple horizontal line (i.e., predicting the mean).

- **Interpretation:**
  - $R^2 = 1$: Perfect prediction.

- $0 \leq R^2 < 1$: The model explains a certain percentage of the variance (e.g., an R^2 of 0.7 means that 70% of the variance in the target data is explained by the model).
- $R^2 < 0$: The model performs worse than predicting the mean of the target variable.

A higher R^2 indicates a better fit, meaning the model is better at explaining the variability of the target data. However, it does not measure how well the model generalizes to unseen data, so it must be evaluated alongside other metrics like RMSE and MAE.

3.7 Limitations

Limited Availability of Long Historical Data APIs One significant limitation encountered in this project was the restricted availability of free APIs that provide long-term historical financial data. Many publicly accessible APIs offer only up to five years of historical data, and retrieving more extensive datasets often requires a paid subscription. This limitation can constrain the depth and reliability of models built on shorter historical data, potentially limiting their ability to capture longer-term market trends and patterns.

In contrast, tools like MetaTrader 5 (MT5) offer the advantage of accessing up to 30 years of historical data for certain assets, making it a valuable resource for developing predictive models that require long-term data for training. The MT5 Terminal, which allows pulling large volumes of historical data, proved crucial in overcoming this limitation. However, the reliance on fewer sources that offer such extensive data can limit the diversity of data inputs and may not cover all asset classes or markets.

Chapter 4

Results and Discussion

4.1 Data Preprocessing

The resulting data frame after preprocessing steps, as outlined in detail in Chapter 3, has the following columns:

Data After Preprocessing (Ready to be Lagged)

- hour_of_day
- minute_of_day
- close
- SMA_10
- EMA_10
- MACD_Histogram
- RSI_14
- Williams_%R

After obtaining this data frame with all the calculations done as above, the following data frame is a result of lagging all the fields by 3 steps:

Lagged Data

- DateTime
- ActualValue
- hour_of_day_lag_3, hour_of_day_lag_2, hour_of_day_lag_1
- minute_of_day_lag_3, minute_of_day_lag_2, minute_of_day_lag_1
- close_lag_3, close_lag_2, close_lag_1
- SMA_10_lag_3, SMA_10_lag_2, SMA_10_lag_1
- EMA_10_lag_3, EMA_10_lag_2, EMA_10_lag_1
- RSI_14_lag_3, RSI_14_lag_2, RSI_14_lag_1
- Williams_%R_lag_3, Williams_%R_lag_2, Williams_%R_lag_1
- MACD_Histogram_lag_3, MACD_Histogram_lag_2, MACD_Histogram_lag_1

4.2 Exploratery Data Analysis

As described in Table 4.1 below, the data directly from the MT5 API has this structure, after it has simply been cast into a data frame type using the Pandas library. The OHLC (Open, High, Low, Close) columns have float values and the time column has date and time data type values. The total number of observations is over 9 million.

The statistics of the data, including mean, standard deviation and so on, are described in Table 4.2 below. The mean of all the OHLC columns are close, with the `open` and the `close` column having the same mean, minimum value, standard deviation, and first, second, and third quartile. All the statistics for all the columns are very close to one another, the difference tends to only be at the fourth decimal place.

TABLE 4.1: DataFrame Structure

Attribute	**Details**
DataFrame Type	pandas.core.frame.DataFrame
RangeIndex	0 to 9294250
Total Entries	9294251
Total Columns	5
Memory Usage	354.5 MB
Column	**Dtype**
time	datetime64[ns]
open	float64
high	float64
low	float64
close	float64

TABLE 4.2: Data Description

Statistic	**time**	**open**	**high**	**low**	**close**
count	9294251	9.29e+06	9.29e+06	9.29e+06	9.29e+06
mean	2012-01-31	1.1851	1.1852	1.1850	1.1851
min	1994-01-03	0.8231	0.8234	0.8227	0.8231
25%	2005-08-11	1.0862	1.0863	1.0861	1.0862
50%	2012-03-08	1.1759	1.1760	1.1758	1.1759
75%	2018-07-02	1.3016	1.3017	1.3015	1.3016
max	2024-09-27	1.6038	1.6039	1.6029	1.6037
std	N/A	0.1542	0.1543	0.1542	0.1542

Each of these columns shares a clearly linear relationship with all other columns, where the plot of all data points forms a straight line with only a few points seemingly just about to step outside of the line (see Figure 4.1). The distribution of all the columns is the same, where the majority of prices lie between 1.0 and 1.2 and the lower extremes on either side being 0.8 and 1.6 (see Figure 4.2). These price ranges are confirmed in Figure 4.3 together with the absence of outliers outside of the range of 0.8 to 1.6.

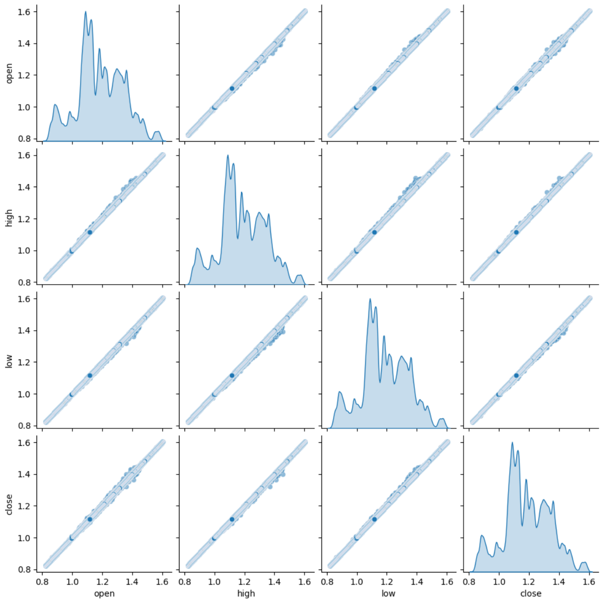


FIGURE 4.1: Pairplot of OHLC data

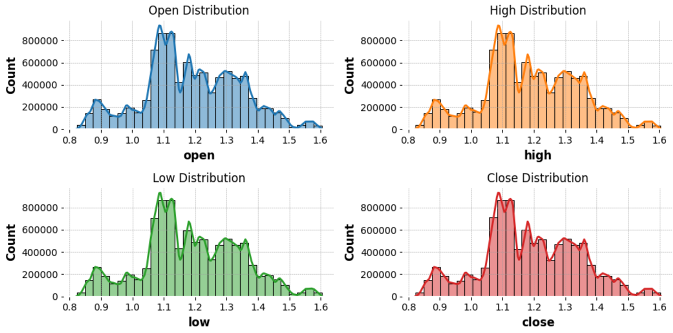


FIGURE 4.2: Distribution of OHLC data

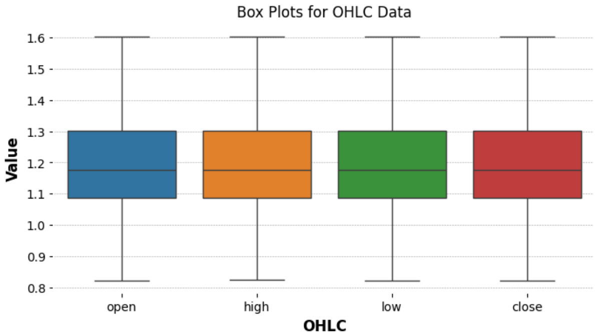


FIGURE 4.3: Distribution of OHLC data (box plot).

The prices for the month of August (chosen at random), show daily trends depending on the time of day (see Figure 4.4). Between the hours of `00h00` and `09h00`, the prices for all days remain fairly constant / fluctuate within a small range. This repeats again between the hours of `22h00` and `24h00`. This pattern is confirmed by Figure 4.5, where average prices for each day of the week are visualized. Price volatility tends to increase between the hours of `09h00` and `22h00`, where peak activity or large dip or surge is around `16h00` for most days.

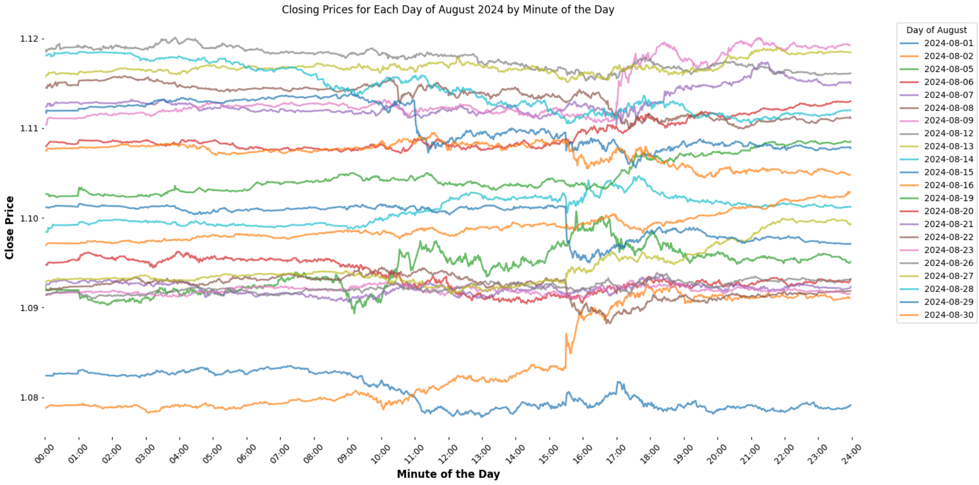


FIGURE 4.4: Price trends per day of the month.

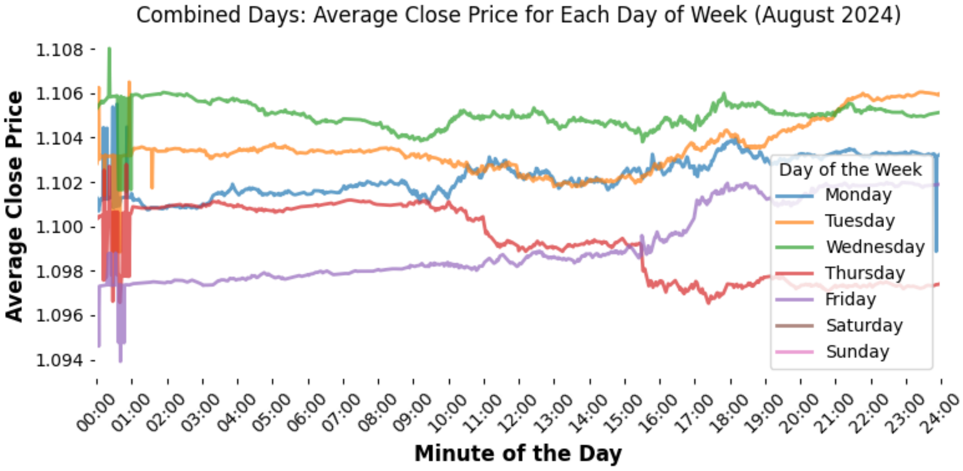


FIGURE 4.5: Price trends per day of the week.

4.3 Results Analysis

4.3.1 Experimental Parameters

Model Architecture

I trained two sequential models, which we shall call **model 1** and **model 2** respectively (see Table 4.3 and 4.4 below). Model 1 was obtained through the random search parameter optimization technique (see Section 4.3.3), whereas model 2 is a structure inspired by the models most used in literature for tasks of similar nature to the one at hand and through the trial and error done in this report to get a general idea of models that perform better. Trial and error show models with fewer (1 or 2) LSTM layers performed better than those higher. Both these make use of Long Short-Term Memory (LSTM) layers. These models differ in complexity, the number of parameters, and their architectures. Both models utilize a combination of LSTM layers, Dropout layers for regularization, and Dense (fully connected) layers for final output prediction. The structure and purpose of each layer are described below.

Model 1 is a deep LSTM-based architecture that came about as a result of hyperparameter optmization. This model contains multiple LSTM layers stacked together, interspersed with Dropout layers to prevent overfitting. The model culminates in a series of fully connected Dense layers, where the final prediction is made. The model architecture as described by the table is as follows.

The column **Layer (type)** lists the layers used in the sequential model, where each layer in the network serves a specific function. The `LSTM` layers are recurrent neural network (RNN) layers capable of learning long-term dependencies. Each LSTM layer can capture temporal dynamics in time-series data, which is key for models dealing with sequential inputs. The `Dropout` layers are used to prevent over-fitting by randomly dropping units during training. This helps the model generalize better. The `Dense` layers are fully connected layers that connect every neuron from the previous layer to every neuron in the current layer. I also use this type of layer for the output of the model.

The **Output Shape** column indicates the dimensions of the data as it flows through each layer. The values `(None, 3, 59)`, which for the first layer, means that

TABLE 4.3: Model 1 Architecture

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 3, 59)	14,396
dropout (Dropout)	(None, 3, 59)	0
lstm_1 (LSTM)	(None, 3, 16)	4,864
dropout_1 (Dropout)	(None, 3, 16)	0
lstm_2 (LSTM)	(None, 3, 62)	19,592
dropout_2 (Dropout)	(None, 3, 62)	0
lstm_3 (LSTM)	(None, 3, 35)	13,720
dropout_3 (Dropout)	(None, 3, 35)	0
lstm_4 (LSTM)	(None, 37)	10,804
dropout_4 (Dropout)	(None, 37)	0
dense (Dense)	(None, 4)	152
dropout_5 (Dropout)	(None, 4)	0
dense_1 (Dense)	(None, 4)	20
dense_2 (Dense)	(None, 1)	5
Total params		**63,553 (248.25 KB)**
Trainable params		**63,553 (248.25 KB)**
Non-trainable params		**0 (0.00 B)**

the first LSTM layer is processing sequences (3 timesteps) of 59 features. "None" refers to the batch size, which is flexible and determined at runtime. The second LSTM layer reduces the feature space to 16 units for each timestep. By the time we reach the final LSTM layer, which has the value `(None, 37)`, the sequence has been flattened, and we have 37 features remaining. The fully connected (Dense) layers, reduce the dimensionality to 4 units, then further reduced to 1 unit (the final output of the model).

The **Param #** column shows the number of parameters (weights) learned by each layer. Dropout layers do not have any parameters because they do not learn weights; they merely control the network's capacity by dropping connections.The values `152, 20, and 5`, for the dense layers refer to the number of trainable weights in the dense layers.

The total parameters `(63,553)` indicate how much the model is learning and how much memory and computation is required for training and inference.

TABLE 4.4: Model 2 Architecture

Layer (type)	**Output Shape**	**Param #**
lstm (LSTM)	(None, 3, 20)	1,760
dropout (Dropout)	(None, 3, 20)	0
lstm_1 (LSTM)	(None, 8)	928
dropout_1 (Dropout)	(None, 8)	0
dense (Dense)	(None, 2)	18
dense_1 (Dense)	(None, 1)	3
Total params		**2,709 (10.58 KB)**
Trainable params		**2,709 (10.58 KB)**
Non-trainable params		**0 (0.00 B)**

Model 2 is a less complex variant of Model 1. It consists of fewer LSTM layers, with a smaller number of units, and the architecture is less deep than Model 1. This model is more lightweight and faster to train. The fields in Table 4.4 for model 2 mean the same as the fields described in detail for 4.3, for model 1.

Hyperparameters

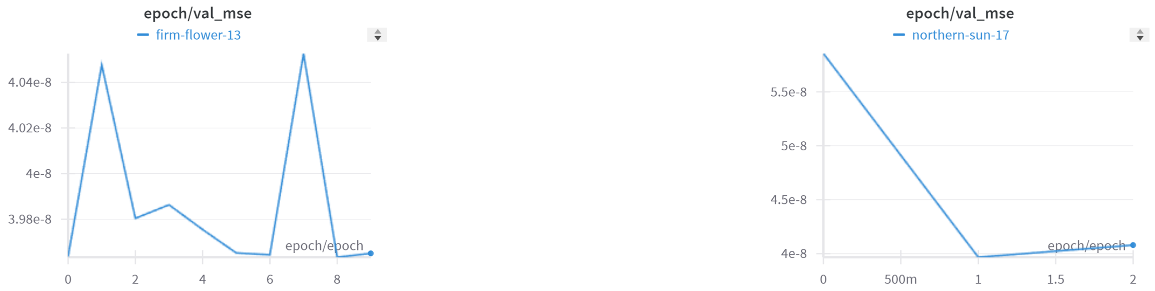
TABLE 4.5: Comparison of Hyperparameters for Model 1 and Model 2

Hyperparameter	**Model 1**	**Model 2**
Number of Layers	5	2
Activation	relu	tanh
Epochs	10	3
Dropout Rate	0.1415	0.1415
Learning Rate	1.5e-05	1.5e-03
Batch Size	32	64
Loss Function	MSE	MSE
Architecture	LSTM	LSTM

4.3.2 Model Training Results

Training and Validation Performance

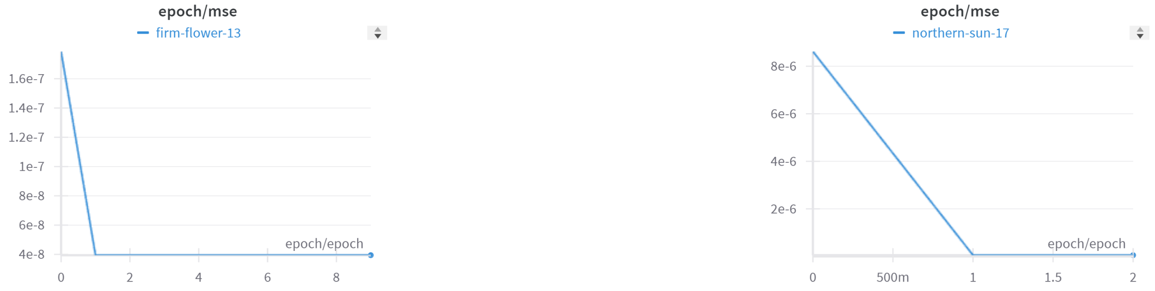
The metrics are visualized per epoch, from Figure 4.6a to Figure 4.12b. In these figures, let ′**(A)**′ represent **model 1** and ′**(B)**′ represent **model 2**, as given in Table 4.5. These metrics tend to behave the same for both models for the training set - they decrease as the epochs go on, up to a point where the graphs appear to plateau, at which point little to no meaningful training (updating of weights) is taking place. For the validation set, taking Figure 4.9b as an example, the loss decreases for model 2 and only begins to rise ever so slightly at epoch 3. For model 1, however, the loss begins by rising and then going down at a slower rate than model 2, up to its lowest point at epoch 6 and it begins to rise again after this point if epochs continue. In instances where this model was trained for more than 10 epochs, during trial and error, the loss oscillated within a clear range for each alternating epochs. In instances where model 2 was trained for more than 3 epochs, the loss, even though it was going up and down, was increasing overall with higher lows each time it attempted to go decrease.



(A)

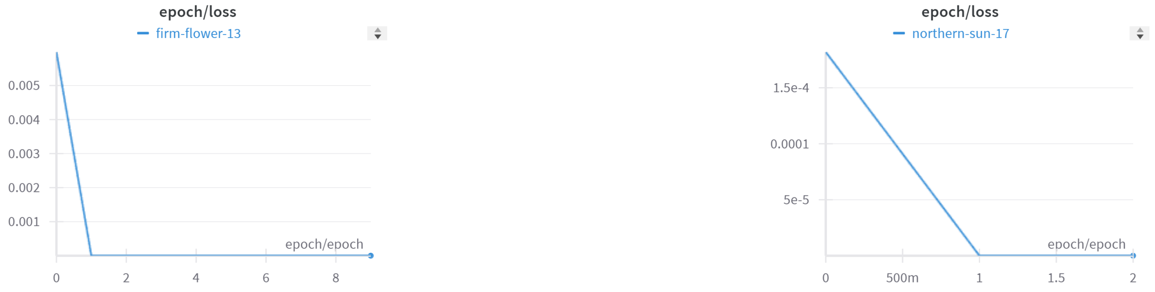
(B)

FIGURE 4.6: Validation MSE comparison between Model 1 and Model 2.



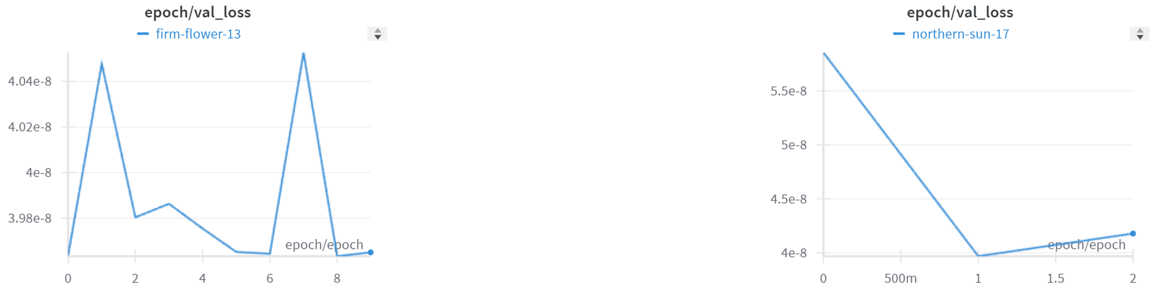
(A) (B)

FIGURE 4.7: Training MSE comparison between Model 1 and Model 2.



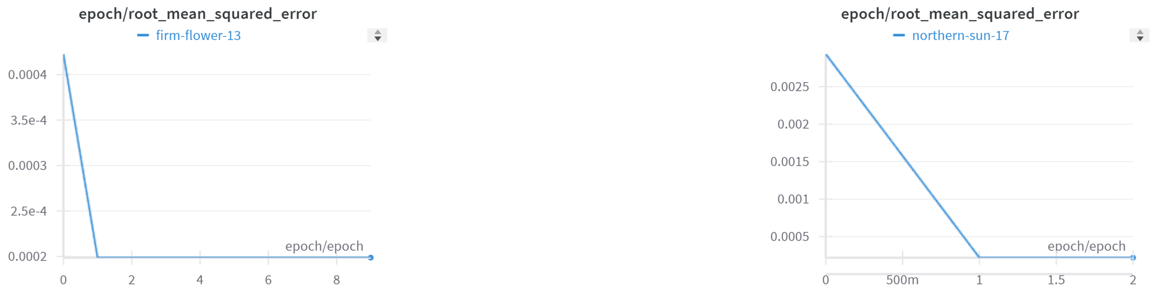
(A) (B)

FIGURE 4.8: Training Loss comparison between Model 1 and Model 2.



(A) (B)

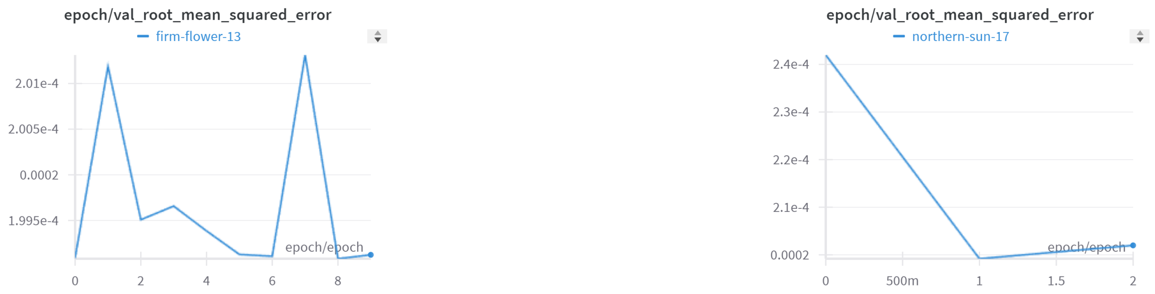
FIGURE 4.9: Validation Loss comparison between Model 1 and Model 2.



(A)

(B)

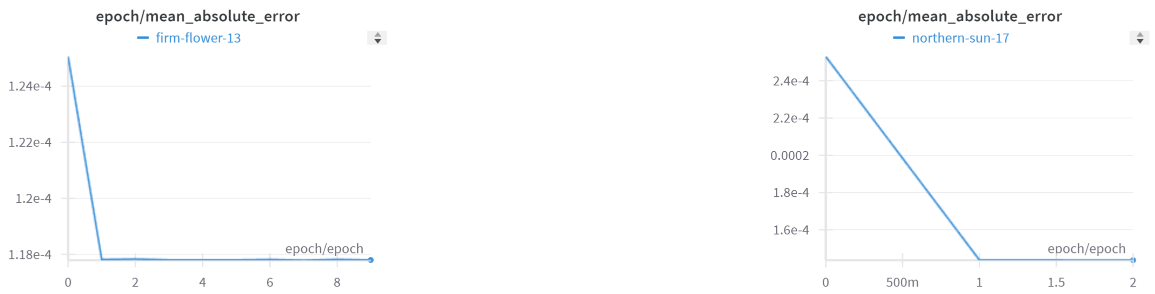
FIGURE 4.10: Training RMSE comparison between Model 1 and Model 2.



(A)

(B)

FIGURE 4.11: Validation RMSE comparison between Model 1 and Model 2.



(A)

(B)

FIGURE 4.12: Training MAE comparison between Model 1 and Model 2.

4.3.3 Hyper-parameter Tuning

The technique for hyper-parameter tuning applied was random search, with 5 trails (see Table 4.3.3). The objective metric to determine the best set of parameters was validation loss - the set of parameters that produced the smallest validation loss would be returned as the best parameters at the end of that trail. Trial 2 (denoted as 1, since the count starts from 0) was returned as the best trial with the smallest validation loss of 3.97e-8. The learning rate came out top in feature importance (see Figure 4.13). The other parameters, in descending order of importance, are the activation function, number of epochs, and dropout rates, with the number of LSTM layers coming out last.

Trial	Value	Activation	Dropout Rate	Epochs	Layer 0 Units	Layer 1 Units	Layer 2 Units	Learning Rate	Num Layers	Layer 3 Units	Layer 4 Units
0	1.37e-7	tanh	0.2274	2	18	25	31	0.03643	3		
1	3.97e-8	relu	0.1415	10	59	16	62	1.51e-5	5	35	37
2	1.88e-6	sigmoid	0.1006	10	29	54	5	0.06682	3		
3	4.79e-8	tanh	0.3434	9	62			0.00136	1		
4	1.57e-7	relu	0.2812	2	59	59	20	0.02034	4	26	

TABLE 4.6: Hyperparameter Optimization Trials

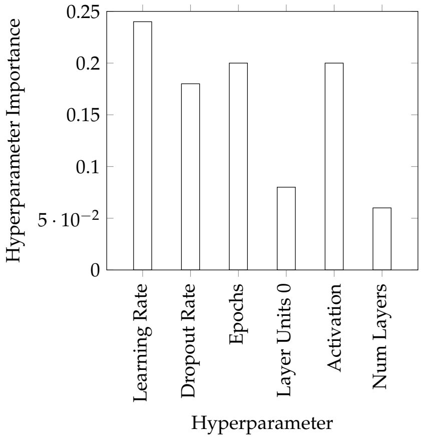


FIGURE 4.13: Hyperparameter Importance from Random Search

4.3.4 Performance Evaluation

Test set evaluation

The performance of the two LSTM models on the test set is summarized in Table 4.7. Model 1 achieved a test loss of 3.9650×10^{-8}, which is lower than the 4.1789×10^{-8} recorded by Model 2, indicating that Model 1 has a better overall fit to the test data. In terms of MAE, Model 1 also outperformed Model 2 with a value of 1.1625×10^{-4} compared to 1.2289×10^{-4}. The RMSE metrics were consistent across both models, with Model 1 yielding 1.9912×10^{-4} and Model 2 yielding 2.0198×10^{-4}.

A metric to truly note, the R-squared values reflect a poor fit for both models, with Model 1 at -4.3094×10^{-4} and Model 2 at -2.9350×10^{-2}, suggesting that neither model explains a significant portion of the variance in the target variable.

Overall, these results indicate that while both models demonstrate relatively low error rates, Model 1 consistently outperforms Model 2 across all evaluated metrics. This suggests that the architecture or training parameters used in Model 1 may be more effective for the specific characteristics of the dataset.

TABLE 4.7: Comparison of Test Results for Model 1 and Model 2

Metric	**Model 1**	**Model 2**
Test Loss	3.9650e-08	4.1789e-08
Mean Absolute Error	1.1625e-04	1.2289e-04
Root Mean Squared Error	1.9912e-04	2.0198e-04
R-squared	-4.3094e-04	-2.9350e-02

4.4 Discussion

Having obtained the results from the models, at first glance, they appear contradictory and inconclusive since the error and loss are very small, and yet the R-squared value is very small too and negative. An R-squared value typically ranges from 0 to 1, where values closer to 1 indicate a better fit of the model to the data, meaning that a higher proportion of variance in the dependent variable is explained by the independent variables. An R-squared value of 0 suggests that the model does not explain any variance, while negative values indicate that the model performs worse than a horizontal line (the mean of the dependent variable). In this case, both models have negative R-squared values, suggesting that they do not adequately capture the relationship between the input features and the target variable. Specifically: Model 1′s R-squared value implies that it explains very little of the variance in the target variable and performs only slightly better than a naive model that predicts the mean of the target variable. Model 2′s R-squared value indicates an even poorer fit, suggesting that it is less effective than Model 1 in explaining variance.

With all that said, the research question posed in this study has been addressed through the development and evaluation of two LSTM-based models. The results indicate that RNN architectures, particularly LSTMs, can be adapted to capture short-term price movements in the forex market, which is characterized by its inherent volatility and rapid fluctuations. The findings from the evaluation of Model 1 and Model 2 support the assertion that LSTM networks can be effective for forecasting forex prices. Model 1 outperformed Model 2 across key metrics, including test loss, MAE, and RMSE, demonstrating that a more complex architecture with optimized hyperparameters can lead to improved prediction accuracy. The significant differences in performance metrics, particularly the lower test loss of Model 1 compared to Model 2′s, suggest that careful tuning of model parameters is crucial for achieving better results in high-frequency trading contexts. Furthermore, despite both models exhibiting poor R-squared values, the overall performance indicates that LSTM networks can still provide valuable insights into short-term price movements.

When comparing these results to existing literature, several studies support the effectiveness of LSTMs and other RNN architectures in financial prediction tasks. For instance, research by [10] demonstrates that LSTMs outperform traditional statistical methods and simpler neural network architectures due to their ability to capture long-term dependencies in time-series data [11]. Additionally, studies have highlighted the importance of feature selection and hyperparameter tuning in enhancing model performance, which aligns with the findings of this research regarding the significance of learning rates and dropout rates [12]. While some research emphasizes hybrid models that combine LSTMs with other machine learning techniques for improved accuracy, this study contributes to the growing body of evidence supporting the use of LSTMs as standalone models for forex price prediction [26]. This is particularly relevant for retail traders seeking to navigate high-frequency trading environments effectively, as the results indicate that a well-optimized LSTM model can provide valuable insights into short-term price movements.

4.5 Summary

Two LSTM-based models were developed to predict forex prices, focusing on optimizing their architectures and hyperparameters. Model 1 utilized a random search technique for optimization, featuring five layers with ReLU activation and a lower learning rate of 1.5e-5. In contrast, Model 2 was based on existing literature and trial-and-error adjustments, employing two layers with a tanh activation function and a higher learning rate of 1.5e-3. Both models incorporated LSTM layers, dropout for regularization, and dense layers for final predictions.

The training process revealed that both models exhibited decreasing loss over epochs, with Model 1 stabilizing after six epochs and achieving a validation loss of 3.97e-8. In the evaluation phase, Model 1 outperformed Model 2 across key metrics: it achieved a lower test loss (3.9650e-8 vs. 4.1789e-8), lower MAE, and slightly better RMSE. However, both models displayed poor R-squared values, indicating limited explanatory power regarding variance in the target variable.

Chapter 5

Conclusion, Recommendations and Future Works

5.1 Conclusion

This study successfully explored the application of RNNs, specifically LSTM architectures, for predicting forex prices in a high-frequency trading environment targeted at retail traders. The research aimed to bridge the knowledge gap regarding HFT strategies and demonstrated that well-optimized LSTM models can effectively capture short-term price movements within a one-minute timeframe. Model 1, with its more complex architecture and optimized hyperparameters, outperformed Model 2 across various performance metrics, highlighting the significance of model design in achieving accurate predictions. The findings speak to the motivation of the research, as it takes another step towards employing machine learning techniques to democratize access to advanced trading strategies, enabling retail traders to compete more effectively in the evolving forex market.

5.2 Recommendations

It is crucial to invest time in hyperparameter tuning and model optimization, as demonstrated by the significant performance differences observed between the two models in this study. So much so in fact, that it is better to dedicate more time in this step than more time performing a trial-and-error exercise. Furthermore, incorporating additional features such as technical indicators or market sentiment data may enhance predictive accuracy. Researchers are encouraged to utilize backtesting frameworks to evaluate model performance and testing by deploying them in live trading environments.

5.3 Future Work

Future research should focus on expanding the scope of this study by exploring hybrid models that combine LSTMs with other machine learning techniques, such as DNNs, ensamble methods or reinforcement learning algorithms. This could lead to improved predictive capabilities by leveraging the strengths of multiple approaches. Another significant gap for future work is exploring real-time implementation of these models in live trading scenarios, assessing their performance under varying market conditions and liquidity levels. Finally, further studies could examine the scalability of these models for different asset classes beyond forex, contributing to a broader understanding of machine learning applications in such financial markets.

Bibliography

[1] Samir Abrol et al. "High frequency trading and US stock market microstructure: a study of interactions between complexities, risks and strategies residing in US equity market microstructure". In: *Financial Markets, Institutions & Instruments* 25.2 (2016), pp. 107–165.

[2] Irene Aldridge. *High-frequency trading: a practical guide to algorithmic strategies and trading systems*. Vol. 604. John Wiley & Sons, 2013.

[3] Monira Aloud and Edward Tsang. "Modelling the trading behaviour in high-frequency markets". In: *2011 3rd Computer Science and Electronic Engineering Conference (CEEC)*. IEEE. 2011, pp. 7–12.

[4] Andrés Arévalo et al. "High-frequency trading strategy based on deep neural networks". In: *International conference on intelligent computing*. Springer. 2016, pp. 424–436.

[5] Jasmina Arifovic, Xue-zhong He, and Lijian Wei. "Machine learning and speed in high-frequency trading". In: *Journal of Economic Dynamics and Control* 139 (2022), p. 104438.

[6] Michael Ayitey Junior et al. "Forex market forecasting using machine learning: Systematic Literature Review and meta-analysis". In: *Journal of Big Data* 10.1 (2023), p. 9.

[7] Svetlana Bryzgalova, Anna Pavlova, and Taisiya Sikorskaya. "Retail trading in options and the rise of the big three wholesalers". In: *The Journal of Finance* 78.6 (2023), pp. 3465–3514.

[8] Gregory W Eaton et al. "Zero-commission individual investors, high frequency traders, and stock market quality". In: *High Frequency Traders, and Stock Market Quality (January 2021)* (2021).

[9] Marinela Adriana Finta, Bart Frijns, and Alireza Tourani-Rad. "Contemporaneous spillover effects between the US and the UK equity markets". In: *Financial Review* 52.1 (2017), pp. 145–166.

[10] Thomas Fischer and Christopher Krauss. "Deep learning with long short-term memory networks for financial market predictions". In: *European journal of operational research* 270.2 (2018), pp. 654–669.

[11] Andreas Gabler et al. "Pattern learning via artificial neural networks for financial market predictions". In: *Available at SSRN 3243479* (2018).

[12] Kirtika Gupta et al. "Prediction of Stock Prices (A Machine Learning Approach)". In: (2021).

[13] Velibor Ilić and Vladimir Brtka. *Evaluation of algorithmic strategies for trading on foreign exchange market*. 2018.

[14] Abdalla Kablan and Wing Lon Ng. "Intraday high-frequency FX trading with adaptive neuro-fuzzy inference systems". In: *International Journal of Financial Markets and Derivatives* 2.1-2 (2011), pp. 68–87.

[15] Robert A Korajczyk and Dermot Murphy. "High-frequency market making to large institutional trades". In: *The Review of Financial Studies* 32.3 (2019), pp. 1034–1067.

[16] Daniel Ladley. "The high frequency trade off between speed and sophistication". In: *Journal of Economic Dynamics and Control* 116 (2020), p. 103912.

[17] Shengnan Li, Edward PK Tsang, and John O'Hara. "Measuring relative volatility in high-frequency data under the directional change approach". In: *Intelligent Systems in Accounting, Finance and Management* 29.2 (2022), pp. 86–102.

[18] L Marudulu. "Portfolio optimisation approaches towards investment in the forex market". PhD thesis. North-West University (South Africa), 2020.

[19] Imad Moosa. "The regulation of high-frequency trading: A pragmatic view". In: *Journal of Banking Regulation* 16 (2015), pp. 72–88.

[20] Peer Nagy, Jan-Peter Calliess, and Stefan Zohren. "Asynchronous Deep Double Dueling Q-learning for trading-signal execution in limit order book markets". In: *Frontiers in Artificial Intelligence* 6 (2023), p. 1151003.

[21] Mattia Pasqual. "High Frequency Trading: an analysis of the phenomenon and the effect of the human component". In: (2020).

[22] ERIK PAULSON et al. "Trading, Investment and Portfolio Management". PhD thesis. WORCESTER POLYTECHNIC INSTITUTE, 2017.

[23] Yuan-Long Peng and Wei-Po Lee. "Data selection to avoid overfitting for foreign exchange intraday trading with machine learning". In: *Applied Soft Computing* 108 (2021), p. 107461.

[24] George Rayment and Michael Kampouridis. "High Frequency Trading with Deep Reinforcement Learning Agents Under a Directional Changes Sampling Framework". In: *2023 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE. 2023, pp. 387–394.

[25] Au Vo and Christopher Yost-Bremm. "A high-frequency algorithmic trading strategy for cryptocurrency". In: *Journal of Computer Information Systems* (2018).

[26] Xuewei Xiang and Weidong Wang. "Predicting Intraday Trading Direction of CSI 300 Based on TCN Model". In: *2023 2nd International Conference on Machine Learning, Cloud Computing and Intelligent Mining (MLCCIM)*. IEEE. 2023, pp. 293–299.

[27] Ailun Ye et al. "Developing sustainable trading strategies using directional changes with high frequency data". In: *2017 IEEE international conference on big data (Big Data)*. IEEE. 2017, pp. 4265–4271.

[28] Khairul Zharif Zaharudin. "A SURVEY OF LITERATURE ON HIGH-FREQUENCY TRADING". In: ().