Multivariate and Multi-Task Deep Learning Architectures for Improved Stock Market Prediction and Risk Management

Osama Assaf

Department of Computer Science

University of Reading

This dissertation is submitted for the degree of Doctor of Philosophy

September 2023

Declaration

I hereby declare that the work presented in this thesis has not been submitted for any other degree or professional qualification and that it is the result of my own independent work.

Osama Assaf

Signed

01/09/2023

Date

Abstract

Effective investment requires adeptly managing trading activities that can adapt to diverse market changes and risks. Key stock market metrics such as volatility, daily returns, and trading volumes play a crucial role in numerous pricing models and trading strategies, including risk management. However, existing tools used to predict these metrics have shown limitations, as evident from recent financial crises. As a result, researchers are examining new methods that leverage machine learning and artificial intelligence to address these weaknesses.

This research makes a significant contribution to the existing body of work on using Deep Learning (DL) for predicting stock market metrics. Different multivariate and multitasking DL architectures are introduced to enhance the prediction accuracy for various future prediction time horizons and different market conditions. For comparison and benchmarking, the models have been assessed against traditional statistical methods and commonly used DL networks.

The results validate the effectiveness of deep learning models in modelling stock market volatility, demonstrating their predictive capability in both upward and downward market conditions across short and long datasets. Furthermore, the study illustrates that deploying multivariate deep learning enhances stock market volatility prediction compared to single-input models. This improvement arises from the utilisation of positively correlated input data and larger datasets, enabling the models to extract crucial information during training and thereby enhancing prediction accuracy.

The deployment of various multi-task deep learning models with shared input layers resulted in significant enhancements in predicting stock market volatility, daily returns, and trading volumes. This improvement stems from the optimisation of the loss function across all output tasks simultaneously. Determining the optimal combination of weights and inputs required an initial step of assigning equal weights to two inputs, followed by an iterative testing process. This iterative testing included the adjustment of the number of inputs and the allocation of weights. The multi-task models demonstrated superior performance relative to both statistical models and single-task deep learning models, including those with multivariate input.

Keywords: Stock Market, Deep Learning, Long Short-Term Memory, Volatility, Returns, Trading Volume, GARCH, ARIMA, Multi-Task Learning, Loss Function

List of Acronyms

AI	Artificial Intelligence
API	Application Programable Interface
ARIMA	Autoregressive Integrated Moving Average
DL	Deep Learning
GARCH	Generalised Auto Regressive Conditional Heteroskedasticity
GRU	Gated Recurrent Unit
LSTM	Long Short-Term Memory
MSE	Mean Squared Error
MTDL	Multi-Task Deep Learning
RMSE	Root Mean Squared Error
RMSProp	Root Mean Square Propagation
RNN	Recurrent Neural Network
STDL	Single Task Deep Learning
VaR	Value at Risk

Publications

Published Papers

Assaf, O., Di Fatta, G., & Nicosia, G. (2022). Multivariate LSTM for Stock Market Volatility Prediction. Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 13164 LNCS, 531–544. https://doi.org/10.1007/978-3-030-95470-3_40/COVER

Submitted Papers

Multi-Task Deep Learning for Stock market Performance Monitoring and Risk Assessment. Assaf, O., Di Fatta, G., Nicosia, G. (Engineering Applications of Artificial Intelligence journal, Submitted in Sep 2023)

Papers in Progress

A comprehensive study of Deep Learning Networks for Forecasting Stock market Volatility. Assaf, O., Di Fatta, G., Nicosia, G. (ICMLC conference, Feb 2024).

Acknowledgements

In acknowledgment of the support and encouragement I have received throughout my Ph.D. journey, I extend my heartfelt gratitude to:

My beloved wife, Fidaa, for her unwavering patience, understanding, and endless support during this challenging yet rewarding endeavour. Your presence has been my anchor, and your belief in me has been my driving force.

My mother, Zuhriyyah, for her boundless love, sacrifices, and unwavering belief in my abilities. Your strength and guidance have shaped me into the person I am today.

My esteemed supervisor, Prof. Giuseppe Di Fatta, for your invaluable guidance, mentorship, and scholarly wisdom. Your expertise and dedication have been instrumental in shaping the course of my research and academic growth.

I also extend my sincere gratitude to my children, Jana, Zaineddin, and Juri, for their unwavering support and patience.

Lastly, I want to express my heartfelt gratitude to my employers at Bank of America for providing me with this invaluable opportunity and for their support throughout my Ph.D. journey.

Table of contents

List of figures

List of tables

List of equations

List of algorithms

Chapter 1

Introduction

In the past few years, particularly following the aftermath of the 2008 financial crisis, the convergence of deep learning and financial markets has become a focal point of considerable interest. The intricate intricacies underlying stock market trends and the pursuit of precise performance forecasting have laid the groundwork for the investigation of inventive methodologies. In this landscape, the integration of deep learning methodologies emerges as a fresh and promising avenue [1].

This research embarks on a compelling journey into the uncharted territory of deep learning for stock market performance prediction, accompanied by a pioneering investigation into the realm of multi-task deep learning. In the ever-evolving landscape of financial markets, where traditional models have faced challenges, this study seeks to unravel the untapped capabilities of these advanced techniques.

The core emphasis of this research centres on three primary goals: firstly, to examine the realm of applying deep learning models to forecast stock market performance; secondly, to examine the potential of utilizing a multivariate deep learning architecture to enhance prediction precision; and thirdly, to investigate the unexamined domain of multi-task deep learning within this context. In the intricate landscape of risk assessment, portfolio management, and precise performance prediction within the financial realm, the integration of deep learning introduces an innovative vantage point.

This topic holds relevance in today financial climate, where the interplay of various market indicators and the rapid pace of information dissemination have rendered traditional methods less effective. By leveraging the power of deep learning, this research aspires to uncover patterns, relationships, and insights that could potentially redefine the way we understand and predict stock market behaviour.

While this study is relevant to various stock market metrics, particular emphasis has been directed towards volatility, returns, and trading volume. This focus arises not only from their pivotal role in portfolio management and overall investment performance, along with their utilization as inputs in numerous pricing and risk assessment models, but also due to their interconnected nature, making them ideal metrics for comprehensive monitoring.

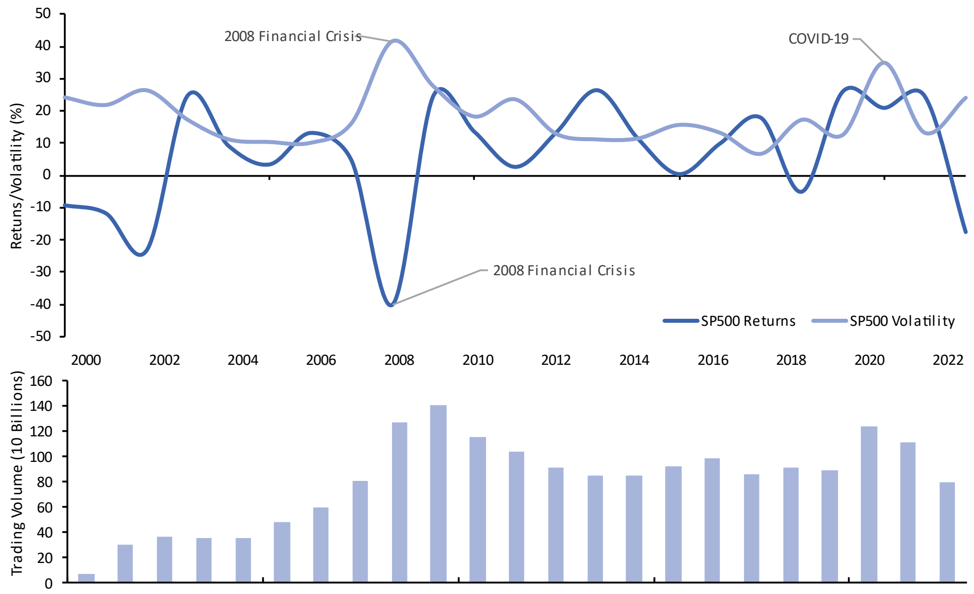


Figure 1.1: S&P 500 volatility, daily returns, and trading volume (2000 – 2022).

As the research embarks on this investigation, it is essential to recognize the novelty of this research. While deep learning has undeniably demonstrated its prowess in various domains, its application to the realm of stock market performance prediction and multi-task learning within this highly dynamic and complex domain remains an emerging and promising avenue. This study marks a significant step in unravelling the untapped potential of deep learning in the financial markets.

This research rigorously examines models and methodologies, seeking to make a substantial contribution to the expanding knowledge at the intersection of deep learning and financial markets. The objective is to uncover new insights in stock market predictions and portfolio monitoring, enhancing precision, insight, and innovation in these domains. This work represents a distinct fusion of advanced technology and financial expertise, with the potential to reshape how market participants understand and navigate the intricate landscape of modern finance and investment banking.

1.1 Problem Statement

The existing methodologies deployed for stock market prediction and risk management have shown ineffectiveness, particularly during financial crises [1], [2]. To address this limitation, there is a pressing need for alternative methods that can either replace or enhance the current tools, enabling faster responses and better management of unforeseen fluctuations in the economy and investments. This research will primarily concentrate on key performance indicators (KPIs) metrics, including volatility, daily returns, and trading volume, aiming to provide improved approaches for forecasting and risk assessment in the financial markets.

1.2 Research Aims and Objectives

The main objective of this PhD research is to examine innovative methods for improving the prediction of key metrics in the stock market, which play a crucial role in trading and risk management platforms.

The research aims to validate three primary hypotheses based on both literature analysis and practical experience in developing quantitative models for mitigating risks associated with the stock market, especially in the context of the recent financial crisis that uncovered significant deficiencies in risk assessment models deployed by financial institutions and investment entities.

Hypothesis 1: The utilization of deep learning models has the potential to enhance or even replace the current models deployed for predicting stock market volatility and optimizing trading portfolios in different market conditions and for different future time horizons.

Hypothesis 2: The efficacy of a multivariate deep learning model surpasses that of a single-input model, attributed to the interrelationship among various input data utilized during the model training phase.

Hypothesis 3: The superiority of a multitasking deep learning model performance compared to a single-task model can be attributed to the integration of a shared layer and a universally distributed loss function across all tasks. Furthermore, altering task combinations and adjusting weights of the loss function can notably influence the precision of predictions.

These hypotheses are grounded in the examination of how deep learning methodologies can address the limitations of existing models and offer innovative solutions to enhance risk management and prediction accuracy in the ever-evolving landscape of financial markets.

Objectives:

1- Perform an extensive literature review on the application of deep learning and multitasking in finance, with a specific focus on its usage for stock market metrics, particularly volatility prediction.
2- Develop innovative Multivariate Deep Learning models designed to improve the predictive accuracy of stock market indicators. These models will serve to assess hypotheses 1 and 2, contrasting their performance against the conventional statistical models currently deployed.
3- Create advanced Multitasking Deep Learning architectures optimized for concurrent prediction of multiple tasks, specifically tailored to enhance the efficiency and precision of forecasting stock market metrics. These models will be pitted against both traditional statistical models and Single-Task Deep Learning models, facilitating the examination of hypothesis 3.
4- Evaluate the newly developed models under various market conditions and different time prediction horizons to ensure their comprehensiveness.
5- Conduct a thorough assessment of the accuracy and performance of the newly developed models in comparison to existing benchmark models, deploying rigorous testing methodologies.
6- Deploy various well-known deep learning networks to compare their performance and suitability concerning all testing parameters.
7- Evaluate the adaptivity of the models to changing data volumes and completeness by conducting tests using both big and small data sets.

The research flow is presented in detail in Diagram 1, which can be found in Appendix A, Section 1.

1.3 Challenges

Model Selection and Validation: Due to the absence of well-established benchmarks, the process of selecting the suitable deep learning architecture and effectively validating the model performance can be complex. Consequently, statistical benchmarks are still deployed.

Challenge 1: **Stock markets exhibit non-stationarity**. Financial markets are dynamic and non-stationary, making it challenging to capture patterns and trends effectively, as past patterns may not repeat in the future [3], [4], [5], [6].

Challenge 2: **High dimensionality**. Stock market data can be high-dimensional, with numerous features, which can lead to overfitting and increased computational complexity [7].

Challenge 3: **Data noise**. Stock market data can be noisy, influenced by external factors, and subject to uncertainty, affecting the accuracy and reliability of predictions [8].

Challenge 4: **Changing market conditions**. Market conditions can change rapidly, and the model performance may degrade when applied to new market situations not seen during training [9], [10].

Challenge 5: **Interpreting model decisions**. Deep learning models are often considered black-box models, making it difficult to interpret their decisions, which can be crucial in financial decision-making [11], [12].

Overcoming these challenges demands meticulous attention to data preprocessing, feature engineering, model architecture selection, hyperparameter tuning, and deploying robust evaluation techniques. It is crucial to acknowledge the limitations and uncertainties inherent in utilizing deep learning for stock market prediction and to augment the models with domain knowledge and risk management strategies.

1.4 Dissertation Contribution

The primary contribution lies in introducing innovative deep learning architectures for predicting stock market volatility and returns. These proposed models offer the potential to enhance portfolio performance and provide valuable insights into the overall market state, thereby facilitating the development of more robust risk management tools that can either integrate or replace existing ones. In the dissertation, various deep learning architectures were experimentally compared to assess their accuracy and speed under diverse market conditions. Additionally, these architectures were juxtaposed with comparable statistical models for further evaluation.

Specifically, the research contributions can be summarized as follows:

Deep Learning Models for Stock Market Volatility Prediction: Defining the architecture of a multivariate LSTM model tailored to predict stock market volatility, showcasing its superiority over traditional statistical models.

Comparison of Single-Input and Multivariate Input Deep Learning Models: Conducting a thorough analysis of stock market indices and prices, examining various recurrent neural network (RNN) models commonly used for time series prediction. The results emphasized the superiority of multivariate deep learning models over single-input ones.

Advancements in Multi-Task Learning: Introducing a multi-task deep learning model that leverages a shared layer across all inputs, leading to enhanced loss functions for all outputs and ultimately improving prediction accuracy.

By focusing on these areas, the research examines the possibility to push the boundaries of deep learning applications in finance, providing valuable insights for portfolio management, risk assessment, and market forecasting.

1.5 Dissertation Outline

The central thesis of this research revolves around the utilization of deep learning algorithms to model volatility, portfolio returns, and other financial time series data with the aim of enhancing trading strategies and risk models to optimize profitability while minimizing risk exposure.

This dissertation presents a comprehensive and practical evaluation of the thesis, supported by experimental testing and analysis of novel and innovative models. Through rigorous experimentation and analysis, the research aims to contribute valuable insights into the application of deep learning in the financial domain, paving the way for improved trading approaches and risk management strategies.

The following is the structure of the thesis:

Chapter 2: **Stock Market and Risk Management**

In chapter 2, the research embarks on an examination of the financial market, trading dynamics, and portfolio management. The research probes extensively into the intricacies of portfolio performance and risk management, particularly focusing on volatility and returns. Additionally, this chapter offers a concise overview of the prevalent statistical models deployed in the current landscape for modelling volatility and time series data.

Chapter 3: **Deep Learning Networks, Loss Function, and Multitasking in Machine Learning**

In chapter 3, The research analyses the inner workings of LSTM and GRU deep learning algorithms, shedding light on their architecture, training setup, tuning methods, and an investigation of their limitations and potential future advancements.

The chapter also conducts an exhaustive study on the techniques deployed for loss computation in regression models, subsequently examining gradient descent and potential challenges encountered in neural networks, particularly in deep learning contexts. Following this, the chapter examines multitasking, shedding light on common architectures and the process of calculating loss functions for models designed to handle multiple tasks.

Chapter 4: **Deep Learning Applications to Stock Market Analysis**

In Chapter 4, a review of the literature regarding machine learning applications in the finance field is conducted. This review places special emphasis on the incorporation of deep learning methods to forecast stock market data. Additionally, an examination of literature concerning the implementation of multitasking to predict multiple stock market metrics is undertaken, along with an investigation of the advantages associated with this approach.

Chapter 5: **Deep Learning for Stock Market Prediction: Unveiling Insights and Enhancing Forecasts**

In Chapter 5, the research addresses one of the primary objectives of this research, focusing on leveraging deep learning to improve the prediction of financial metrics. Additionally, the research introduces an innovative deep learning model utilizing multivariate LSTM to forecast stock market asset volatility.

Chapter 6: **Multitasking in Trading Markets: Enhancing Decision-Making with Multi-Task Deep Learning**

Chapter 6 undertakes a comprehensive examination of another crucial research objective which revolves around the introduction of a novel multi-task deep learning architecture for modelling stock market metrics, including daily returns, trading volume, and the critical market performance indicator - volatility.

Chapter 7: **Concluding Remarks and Future Directions**

In Chapter 7, the research draws conclusions from this dissertation, summarizing the main contributions made throughout the research journey. Furthermore, the research outlines potential areas for future research and improvements to the proposed thesis.

Chapter 2

Stock Market and Risk Management

Chapter 2 will focus on introducing the key concepts and topics that will be analysed throughout the study. In this chapter, the following topics are covered:

Portfolio Management: This section provides an overview of portfolio management, its importance in finance, and the various strategies used to construct and manage investment portfolios.

Stock Market: Within this section, the concept of the stock market and the act of trading are expounded upon. The reader is introduced to the overarching characteristics that define the stock market at a broader level.

Risk Management: This section explains the concept of risk management in finance and how it involves identifying and mitigating risks to achieve financial objectives.

Volatility: This section defines volatility as a measure of price fluctuation in financial assets and discusses its importance in risk assessment and pricing models.

Daily Returns: This section defines daily returns as a financial metric that measures the percentage change in the price of an asset from one trading day to the next. It may discuss how daily returns are calculated and their significance in evaluating investment performance.

Trading Volume: This section introduces trading volume as a metric that measures the total number of shares or contracts traded during a specified time period. It examines how trading volume reflects market interest and liquidity.

Statistical Models: This section provides an overview of statistical models commonly used in finance for forecasting and prediction. It may include an introduction to models such as ARIMA (Autoregressive Integrated Moving Average) or GARCH (Generalized Autoregressive Conditional Heteroskedasticity).

In Chapter 2, the aim is to lay the foundation for the study, familiarize readers with key financial concepts, and set the stage for the subsequent chapters that investigate each topic and examine and analyse their relationships and applications in the context of the research.

2.1 Stock Market

The stock market, often referred to as the equity market, is a vital component of the global financial system where shares of publicly traded companies are bought and sold. It functions as a marketplace where investors, such as individuals and institutions, can purchase ownership stakes, or "shares," in companies. These shares represent a portion of the company ownership and entitle the shareholder to a portion of its profits and potential growth [10].

Outlined below are the pivotal characteristics of the stock market along with essential terminology.

Initial Public Offerings (IPOs) take place in the *primary market*. Companies issue new shares to the public for the first time, raising capital to fund their operations and expansion. IPOs are commonly orchestrated by financial institutions through a process known as underwriting, during which the bank assesses the company that is presenting shares (issuer) and facilitates the availability of these shares for public purchase [10], [13].

A *share* refers to an individual unit of ownership in a company. Shares represent the division of ownership of a company equity among its investors. Shareholders may have certain rights, such as voting on major corporate decisions and receiving dividends. Shares can be of different types, such as common shares and preferred shares, each with its own set of rights and characteristics [13].

After shares are vended to the public in the primary market, they become eligible for trading on the *secondary market*, commonly recognized as the stock market. Here, investors engage in the exchange of shares amongst themselves.

The *stock* is a broader term that encompasses the entirety of ownership units in a company. It refers to the total capital raised by a company through the issuance of shares. In other words, stocks represent the collection of all shares that are issued by a company [10], [13]. Stocks can be held by multiple shareholders, and they are traded on stock exchanges. The term "stock" is often used interchangeably with "equity" and is used to indicate the ownership interest of investors in a company.

A *stock option* is a financial contract that gives the holder the right, but not the obligation, to buy or sell a specific number of shares of a company stock at a predetermined price, known as the "strike price," within a specified period, typically referred to as the "expiration date" or "maturity date." There are two primary types of stock options, *call option* and *put option* [5], [14], [15].

A call option gives the holder the right to buy shares of stock at the strike price. Call options are often used by investors who believe the stock price will rise, enabling them to purchase shares at a lower price than the market value.

A put option gives the holder the right to sell shares of stock at the strike price. Put options are typically used by investors who anticipate that the stock price will fall, enabling them to sell shares at a higher price than the market value.

Options are valued through various mathematical models, with one of the most renowned being the Black-Scholes formula.

The application of multivariate deep learning research, as detailed in **Chapter 5**, entails forecasting the performance of Bank of America (BAC) stock while incorporating additional features or inputs, such as JP Morgan Chase (JPM) stock price, City Bank (C) stock price, and Crude Oil prices.

Stock exchanges, such as the New York Stock Exchange (NYSE), Nasdaq, London Stock Exchange (LSE) provide a regulated platform for trading stocks. These exchanges set rules and facilitate the buying and selling of shares [10], [13].

Stock market indices, such as the S&P 500 or Dow Jones Industrial Average, track the performance of a specific group of stocks, providing a snapshot of overall market health and trends. In **Chapter 6**, the primary focus of investigation revolves around these two indices, serving as the central study cases for the multitasking deep learning research.

The stock market can be subject to significant price fluctuations due to various factors such as economic indicators, company performance, geopolitical events, and investor sentiment. Such changes in price are referred to as *volatility* [9].

Changes to stock prices are referred to as *returns*, which can be monitored on different time windows, The number of shares traded on the stock market is referred to as t*rading volume*.

Volatility, daily returns, and trading volumes are explained in the coming sections of this chapter.

Individuals, institutions, and even governments participate in the stock market as *investors*. They buy shares with the expectation of earning a return on their investment through capital appreciation (increased share value) or dividends (a portion of the company profits).

In addition to investors, the stock market involves various other participants such as *brokers*, market makers, *regulators*, and *financial analysts* who contribute to the functioning and regulation of the market [10], [13].

Investors and analysts deploy diverse approaches, encompassing fundamental analysis and technical analysis, for assessing companies and making well-informed choices regarding stock transactions. This plays a crucial role in sustaining profitable investments. Consequently, various pricing models, risk assessment methodologies, and portfolio management strategies are embraced by entities participating in these activities. Furthermore, vigilant monitoring and regulation of these practices are overseen by the financial authorities of each respective country [13].

The stock market plays a crucial role in allocating capital and facilitating economic growth. It provides companies with a means to raise funds for expansion and innovation while offering individuals and institutions opportunities to invest and grow their wealth. The dynamics of the stock market are influenced by a complex interplay of economic, financial, and psychological factors, making it a dynamic and ever-evolving arena [13].

2.2 Portfolio Management

Portfolio management involves the process of selecting and managing a combination of assets, known as a portfolio, to achieve specific financial goals and objectives [10]. Three important performance measures commonly used in portfolio management are the Sharpe ratio, the Treynor Measure, and the Jensen Measure.

These three performance measures are valuable tools for investors and portfolio managers to evaluate the risk-adjusted returns of a portfolio and make informed decisions in managing investments. Each measure provides a different perspective on the portfolio performance and risk characteristics, helping investors to optimize their investment strategies [1], [10].

2.2.1 Sharpe Ratio

The Sharpe ratio is a risk-adjusted measure that assesses the excess return of a portfolio per unit of risk taken. It was developed by Nobel laureate William F. Sharpe [10], [16].

The formula for the Sharpe ratio is:

$$Sharpe\ Ratio = \frac{R_p - R_f}{\sigma_p}$$

Equation 2.1: Sharpe ratio formula.

Where:

R_p is the portfolio average rate of return,

R_f is the risk-free rate of return (e.g., government bond yield),

σ_p is the standard deviation of the portfolio returns.

The Sharpe ratio quantifies how much additional return an investor receives for each unit of risk assumed. A higher Sharpe ratio indicates a more favourable risk-to-reward profile for the portfolio.

2.2.2 Treynor Measure

The Treynor Measure, developed by Jack L. Treynor, is another risk-adjusted performance measure used in portfolio management [10], [16]. It evaluates the portfolio excess return per unit of systematic risk. The formula for the Treynor Measure is:

$$Treynor\ Measure = \frac{R_p - R_f}{\beta_p}$$

Equation 2.2: Treynor measure formula.

Where:

R_p is the portfolio average rate of return,

R_f is the risk-free rate of return (e.g., government bond yield),

β_p is the portfolio beta, a measure of systematic risk (covariance of portfolio returns with the market divided by the market variance).

The Treynor Measure helps investors assess how well the portfolio manager has performed relative to the systematic risk taken.

2.2.3 Jensen Measure (Jensen Alpha)

The Jensen Measure, also known as Jensen Alpha, is a risk-adjusted performance measure that evaluates a portfolio performance relative to its expected return based on the Capital Asset Pricing Model (CAPM) [10], [16]. It was developed by Michael C. Jensen. The formula for the Jensen Measure is:

$$Jensen\,Measure\; = R_p - (R_f + \beta_p \times (R_m - R_f))$$

Equation 2.3: Jensen measure formula.

Where:

R_p is the portfolio average rate of return,

R_f is the risk-free rate of return,

β_p is the portfolio beta,

R_m is the average rate of return of the market.

The Jensen Measure assesses whether the portfolio returns have outperformed or underperformed what would be expected given its risk exposure according to the CAPM.

2.3 Risk Management

Risk management in finance refers to the process of identifying, assessing, and mitigating potential risks that could adversely affect the financial health of an individual, organization, or investment portfolio. It involves the implementation of strategies and techniques to minimize the impact of uncertain events on financial outcomes.

In the context of finance, risk can arise from various sources, including market fluctuations, credit defaults, interest rate changes, exchange rate fluctuations, and operational issues, among others. Effective risk management aims to protect assets, preserve capital, and ensure the stability and sustainability of financial operations.

The first step in risk management is identifying potential risks and understanding their nature and potential impact on financial performance. This involves analysing market trends, economic conditions, industry-specific risks, and internal factors.

Once risks are identified, they need to be assessed in terms of their probability of occurrence and potential severity. Quantitative methods, such as Value at Risk (VaR) and stress testing, may be used to measure the extent of potential losses [14], [17].

After assessing risks, strategies are developed to mitigate their impact. This may include diversification of investments, hedging through derivative instruments, using insurance products, or implementing operational controls and safeguards.

Risk management is an ongoing process that requires continuous monitoring and evaluation of the effectiveness of risk mitigation strategies. Regular assessments are made to determine if any adjustments or improvements are necessary.

Financial institutions and investors have specific risk tolerance levels, which represent the amount of risk they are willing to accept. Risk appetite defines the level of risk they are willing to take to achieve their financial objectives.

In the financial industry, there are often regulatory requirements related to risk management. Financial institutions and investment funds must adhere to these regulations to ensure compliance and avoid potential penalties.

Risk management is a critical aspect of financial decision-making, as it helps investors and financial institutions to make informed choices, optimize returns, and protect against potential losses. By effectively managing risks, individuals and organizations can maintain financial stability and enhance their ability to achieve their financial goals.

2.3.1 Portfolio Risk

Portfolio risk refers to the uncertainty or variability of returns associated with holding a collection of investments, known as a portfolio. When an investor combines multiple assets in their portfolio, the overall risk of the portfolio is not just the sum of the individual risks of each asset. Instead, it is influenced by the correlation or relationship between the assets.

Diversification is a key strategy to manage portfolio risk. By combining assets that have low or negative correlations with each other, investors can reduce the overall risk of the portfolio. This is because when some assets in the portfolio may perform poorly, others may perform well, balancing out the overall returns [3], [10].

The two main types of portfolio risk are systematic risk (also known as market risk) and unsystematic risk (also known as specific risk).

Systematic risk affects the entire market or a broad segment of it. It is caused by factors that are beyond the control of individual investors, such as changes in interest rates, economic conditions, political events, and market sentiment. Systematic risk cannot be eliminated through diversification since it is inherent in the overall market.

Unsystematic is specific to individual assets or industries and can be reduced through diversification. Unsystematic risk includes risks related to the performance of specific companies, management issues, industry-specific factors, and other idiosyncratic events. By holding a diversified portfolio with a mix of assets from different industries and sectors, investors can mitigate unsystematic risk.

The total risk of a portfolio is the combination of systematic risk and unsystematic risk. To measure and manage portfolio risk, investors use various risk metrics such as standard deviation, beta, Sharpe ratio, and Value at Risk (VaR). By understanding and managing portfolio risk, investors aim to achieve their financial objectives while balancing risk and return according to their risk tolerance and investment goals.

2.3.2 Value at Risk (VaR)

Value at Risk (VaR) is a statistical measure used to estimate the potential loss, at a specific confidence level, in the value of an investment or portfolio over a given time horizon. VaR is commonly used in finance to assess the risk of investments and portfolios and is a key tool in risk management [14], [18], [19].

VaR tackles the question of determining the highest possible loss within a specified timeframe with a certain level of confidence. For instance, when the one-day 95% VaR is $1 million, it indicates a 95% probability that the investment or portfolio will not experience a loss exceeding $1 million in the next day.

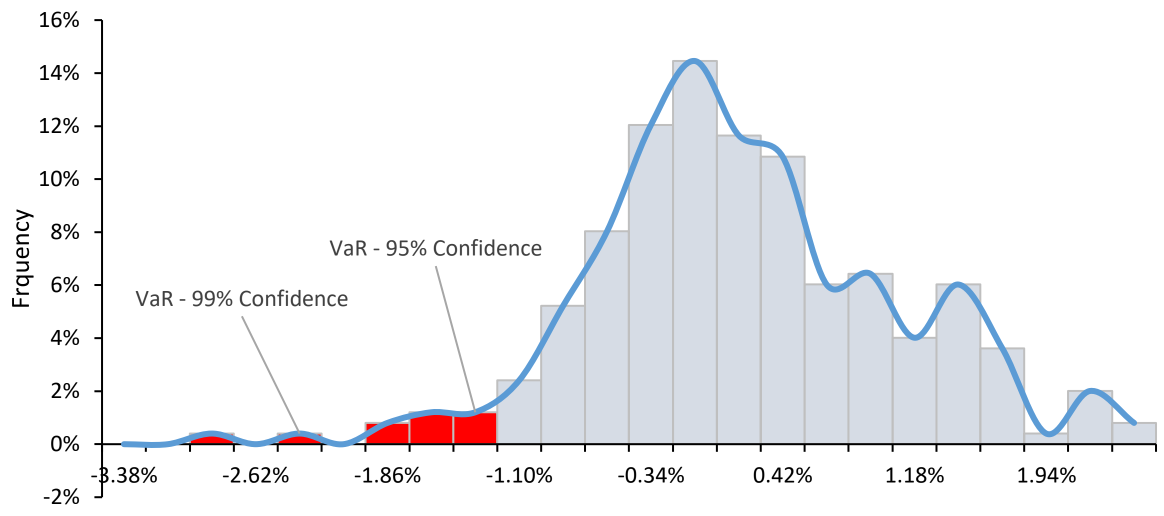


Figure 2.1: Value at Risk (VaR) – JP Morgan Chase (2017).

To calculate VaR, historical data on the returns of the investment or portfolio are used to construct a probability distribution of potential future returns. The confidence level chosen determines the percentile of the distribution used to estimate VaR. For example, a 95% VaR uses the 5th percentile of the distribution, which represents the value below which 95% of the potential losses fall.

VaR takes into account the volatility of the investment or portfolio, as well as the correlations between different assets if it is a portfolio. It provides a concise and standardized measure of risk, enabling investors and financial institutions to compare the riskiness of different investments or portfolios.

However, it is important to note that VaR has limitations. VaR provides a point estimate of potential losses but does not capture the potential severity of extreme losses beyond the specified confidence level. Additionally, VaR assumes that the distribution of future returns will be similar to historical data, which may not always hold true during times of market stress or extreme events [18].

2.4 Volatility

Volatility is a metric that gauges the level of fluctuation in the price of a security or a market index. It serves as a fundamental indicator of risk and provides valuable insights for effectively managing uncertainty in the financial markets [20].

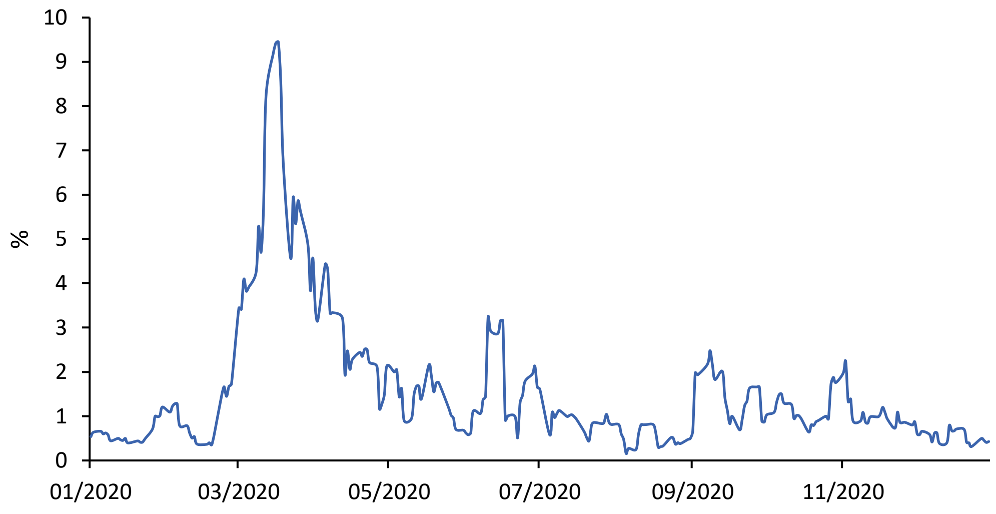


Figure 2.2: S&P 500 index volatility due to COVID-19 – 2020.

2.4.1 Historical Volatility

Volatility holds significant importance in trading and risk management, and various methods exist to calculate or derive it. Historical Volatility relies on the time series of historical prices for the securities within a portfolio to calculate returns [20], [21]. The return for each asset is computed independently using this approach. The average return R_{avg} is defined as:

$$R_{avg} = \frac{\sum_{i=1}^{n} Ri}{n}$$

Equation 2.4: Average daily returns formula.

Where:

n is the sample size of the time-series.

Ri is the security return.

The volatility σ_{Sec} of a single asset and the Annualized Volatility σ_{An} are given by the following formulas.

$$\sigma_{Sec} = \sqrt{\frac{\sum_{i=1}^{n} \left(R_i + R_{avg}\right)^2}{n-1}}$$

Equation 2.5: Volatility (Standard-Deviation) formula.

$$\sigma_{An} = \sigma_{Sec}\sqrt{n}$$

Equation 2.6: Volatility annualised formula.

To calculate volatility for a portfolio comprising multiple assets, it is essential to consider the correlation coefficient and the respective weights of each asset. These parameters play a crucial role in determining the overall volatility of the portfolio.

For example, the volatility σ_p of a two-asset portfolio is calculated by the following formula.

$$\sigma_p = \sqrt{W_1^2 \sigma_1^2 + W_2^2 \sigma_2^2 + 2w_1 W_2 \sigma_1 \sigma_2 \rho_{1,2}}$$

Equation 2.7: Volatility of two assets portfolio formula.

Where:

W is the proportion of each asset in relation to the whole portfolio.

σ_1 is volatility for asset 1 in the portfolio.

σ_2 is volatility for asset 1 in the portfolio.

$\rho_{1,2}$ is the correlation coefficient of assets 1 and 2.

For portfolios comprising more than two assets, the general formula below can be used to represent the volatility.

$$\sigma_p^2 = \begin{bmatrix} w_1 & \cdots & w_n \end{bmatrix} \begin{bmatrix} \sigma_{11} & \cdots & \sigma_{1n} \\ \vdots & & \vdots \\ \sigma_{n1} & \cdots & \sigma_{nn} \end{bmatrix} \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix}$$

Equation 2.8: Volatility of multiple assets portfolio formula.

2.4.2 Implied Volatility

Implied volatility is a measure used in options trading to assess the market expectations of future price fluctuations for an underlying asset. It is a critical component in determining the price of options contracts.

Options are financial derivatives that give the holder the right, but not the obligation, to buy or sell an underlying asset at a predetermined price (known as the strike price) on or before a specified expiration date. Implied volatility is inferred from the market price of options, as it reflects the market participants' collective view on the potential magnitude of price movements for the underlying asset.

When implied volatility is high, it suggests that market participants anticipate significant price fluctuations in the underlying asset. Conversely, when implied volatility is low, it indicates that the market expects relatively stable price movements for the underlying asset.

Traders and investors use implied volatility to gauge the market sentiment and make informed decisions regarding options trading strategies. It can also be used to assess the relative expensiveness or cheapness of options contracts. High implied volatility may lead to higher option premiums, while low implied volatility may result in lower option premiums.

Implied volatility is an important parameter in option pricing models, such as the Black-Scholes model [15], [20], [17] which use it to calculate theoretical option prices based on other factors such as the current market price of the underlying asset, the option strike price, time to expiration, and risk-free interest rate. The formula of Black-Scholes can be seen below.

$$d_1 = \frac{\ln\left(\frac{S_0}{K}\right) + \left(r + \frac{\sigma^2}{2}\right)T}{\sigma\sqrt{T}}$$

$$d_2 = d_1 - \sigma\sqrt{T}$$

$$c \ = S_0 N(d_1) - Ke^{-rT} N(d_2)$$

$$p \ = Ke^{-rT} N(-d_2) - S_0 N(-d_1)$$

Equation 2.9: Black Scholes option pricing formula.

Where:

S_0 is security price.

K is the strike price.

r is interest free rate.

T is time to expiration in years.

σ is volatility.

N () is normal distribution.

c is the call option price.

p is put option price.

2.4.3 Beta (β)

Market beta, also known as beta coefficient or beta, is a measure of a stock or a portfolio sensitivity to movements in the overall market or a specific benchmark index. It quantifies the relationship between the returns of the individual asset or portfolio and the returns of the market as a whole.

The market beta is calculated through regression analysis, where historical data of the asset or portfolio returns is compared to the returns of the market index [10], [21]. The beta coefficient is the slope of the regression line, and it represents the change in the asset or portfolio returns for a one-unit change in the market returns.

A beta value of 1 indicates that the asset or portfolio tends to move in line with the market. A beta greater than 1 implies that the asset or portfolio is more volatile than the market, and its price tends to move more than the market in either direction. A beta less than 1 indicates that the asset or portfolio is less volatile than the market, and its price tends to be more stable than the market price.

Interpretation of beta:

Beta = 1: The asset or portfolio moves in line with the market.

Beta > 1: The asset or portfolio is more volatile than the market.

Beta < 1: The asset or portfolio is less volatile than the market.

Beta = 0: The asset or portfolio has no correlation with the market; hence, the asset of portfolio is risk free.

Investors use beta as a risk measure to assess how much an asset or portfolio returns are influenced by market movements. A high beta indicates higher volatility and risk, while a low beta suggests lower volatility and risk. Beta is an important tool in portfolio management, as it helps investors determine how the asset or portfolio behaves relative to the overall market and enables them to adjust their portfolios based on their risk tolerance and investment objectives.

2.4.4 Volatility and Options Pricing

Volatility plays a crucial role in option pricing. In the context of financial derivatives such as options, volatility refers to the degree of variation or fluctuation in the price of the underlying asset. It is a measure of market uncertainty and represents the market expectations about the future price movements of the asset.

The Black-Scholes model is one of the most widely used models for pricing European-style options (options that can only be exercised at expiration). This model assumes that the underlying asset price follows a geometric Brownian motion, and it considers factors such as the current price of the asset, the option strike price, the time to expiration, the risk-free interest rate, and the asset volatility [10], [20], [21], [22]. Volatility is a critical component in the Black-Scholes formula as it directly affects the option price. Higher volatility generally results in higher option prices, as it increases the probability of large price swings and potential profit opportunities.

Implied volatility is the market expectation of future volatility, derived from the current option prices. It is a forward-looking measure that represents the collective opinion of market participants about the asset future price movements. Traders and investors use implied volatility to gauge the market sentiment and make informed decisions about options trading strategies. If the implied volatility is high, it suggests that market participants anticipate significant price swings, leading to higher option prices. Conversely, low implied volatility indicates expected stability in the asset price, resulting in lower option prices.

Volatility is an essential input in option pricing models, hence, changes in volatility can significantly impact option prices. Traders and investors closely monitor volatility levels and make adjustments to their option strategies accordingly to take advantage of market conditions. Additionally, implied volatility can be used to identify mispriced options and opportunities for option trading strategies, such as volatility trading and volatility arbitrage.

2.5 Daily Returns

The Profit and Loss (PnL) on an asset is defined as the change in price of the asset over a specific period of time [5], [10]. Positive returns occur when the new price at time T is greater than the price at time T-1, whereas negative returns occur when the price at time T is smaller than the price at time T-1. In many financial applications, the rate of returns is calculated by measuring the change between two prices from different times, which can be achieved using either arithmetic or log methods. Arithmetic Returns, R_{arth} and Log Returns R_{log} for asset prices $Price_{t-1}$ and $Price_t$ be calculated using the formulas below:

$$R_{arth} = \frac{Price_t - Price_{t-1}}{Price_{t-1}}$$

Equation 2.10: Daily Returns – Arithmetic formula.

$$R_{log} = \frac{\log(Price_{t-1})}{\log(Price_t)}$$

Equation 2.11: Daily Returns – Logarithmic formula.

Daily returns represent the changes in an asset price on a daily basis, and as a result, annual returns are derived by summing all the daily returns over the course of a year. The same principle is applied to calculate returns for other custom periods, such as weekly or quarterly returns.

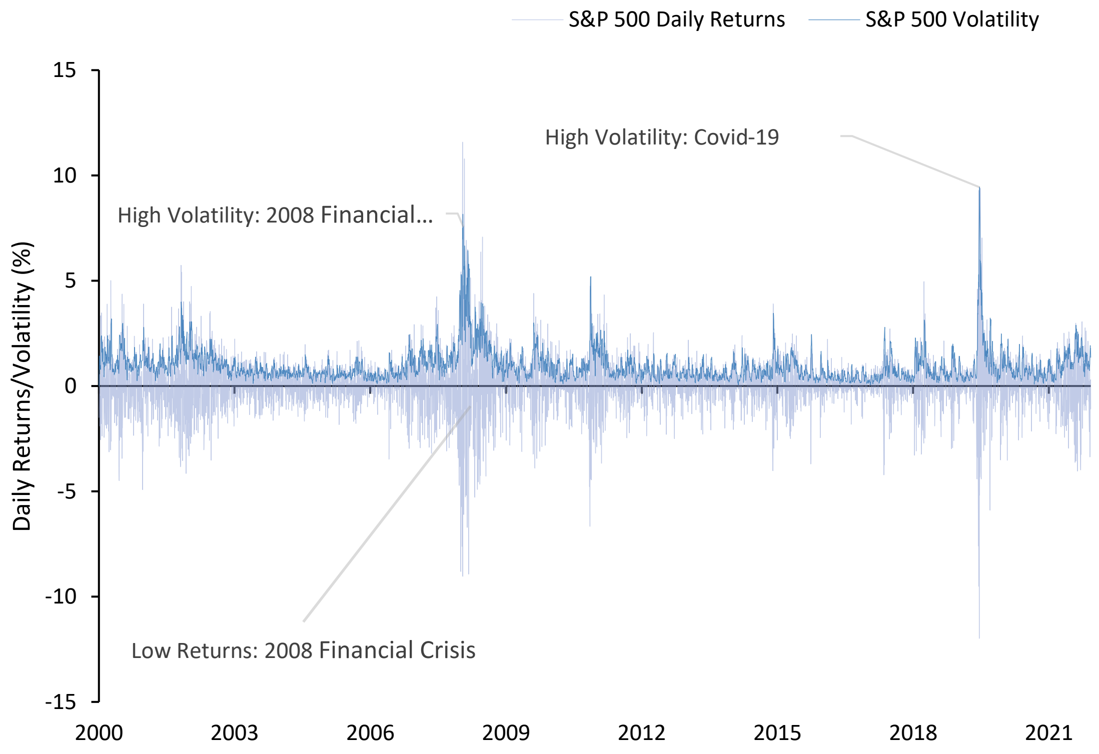


Figure 2.3: S&P 500 volatility and daily returns 2000 – 2022.

In general, investors are always seeking assets that offer positive returns. The potential for positive returns largely depends on the performance of the asset during the holding period. For instance, some investors may engage in short-term trading, buying and selling assets within the same day to capitalize on immediate profits. Conversely, other investors may opt for longer holding periods, spanning days, weeks, months, or even years, based on their forecasts of how the asset returns will evolve during the holding period. The decision to hold an asset for a specific duration is influenced by their expectations of its price movements and potential for favourable returns.

Positive returns in the market are not always guaranteed, and not all traders are solely interested in bidding on stocks to go up. Therefore, accurate forecasting of returns and overall market performance is essential for formulating a profitable investment strategy. Additionally, considering other metrics such as trading volumes and volatility can further enhance the precision of returns forecasting.

2.6 Trading Volume

Trading volume refers to the total contracts or shares traded during a period of time which is usually measured on a daily basis and accumulated on different horizons such as weekly, monthly, yearly, or quarterly. Increase or decrease in trading volume generally reflects the supply and demand for a particular asset, however, significant changes in volumes usually indicates market events such as financial crises and global events such as wars and pandemics. It is a significant metric that reflects the level of activity and liquidity in a particular market or security.

Trading volume can be used to confirm price trends which is essential for technical analysis, it can also be used to identify potential trend reversals or continuations and an essential indication of liquidity. It can also be used by traders to know the precise time to enter or exit a trading activity.

While high trading volumes can generally indicate an aggressive bull market, it can also be an indication of financial crises, hence, risk management systems include it to monitor any trends in the market that could indicate a potential risk to trading portfolios.

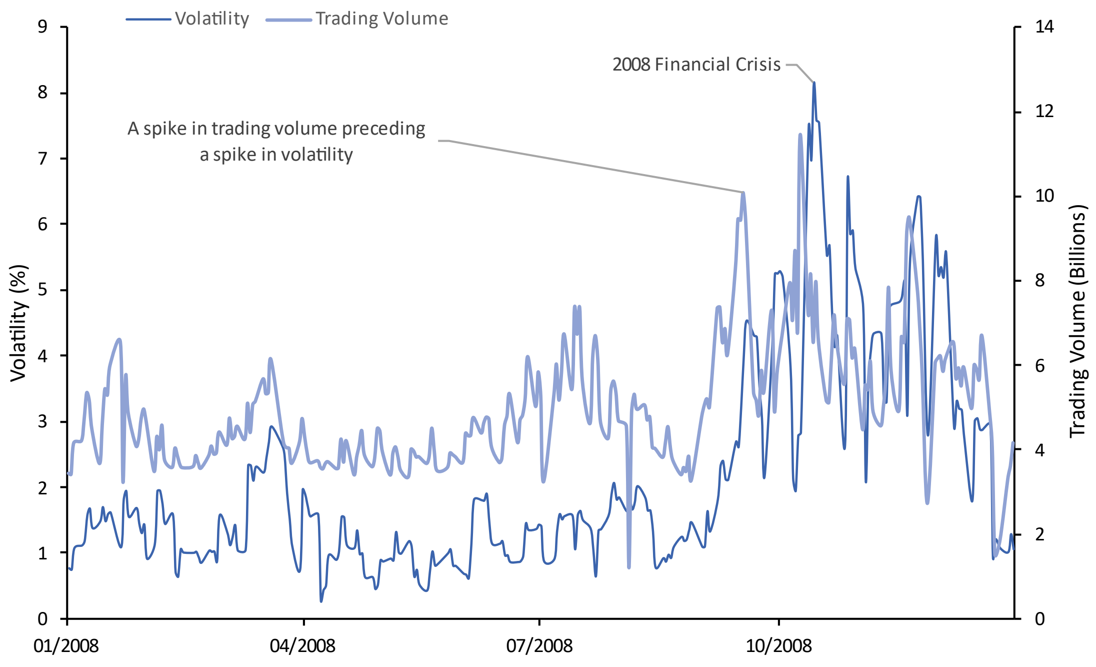


Figure 2.4: S&P 500 volatility and trading volume, 2008.

In general, trading volume is an essential tool for traders to assess market dynamics, identify trends, gauge market sentiment, and make informed trading decisions.

Figure 2.1 depicts the observed trend between trading volume and the volatility spike in the S&P 500 during the 2008 financial crisis, with trading volume showing notable patterns days before the volatility surge.

2.7 Statistical Models

GARCH and ARIMA models are fundamental tools in time series analysis, with GARCH specializing in modelling volatility for risk assessment, while ARIMA provides a robust framework for forecasting and modelling various types of time series data by addressing non-stationarity and incorporating past values and forecast errors. These models have had a significant impact on fields were understanding and predicting time-dependent data is crucial.

2.7.1 GARCH for Volatility Modelling

The Generalized Autoregressive Conditional Heteroskedasticity (GARCH) model is a statistical model commonly used to analyse and forecast the volatility of financial time series data, such as stock prices, exchange rates, or commodity prices. It was introduced by Robert Engle in 1982 as an extension of the Autoregressive Conditional Heteroskedasticity (ARCH) model proposed by Robert F. Engle and Clive Granger in 1982 [23], [24].

The GARCH model is designed to capture the time-varying volatility or variance clustering present in financial time series. Volatility clustering refers to the tendency of financial returns to exhibit periods of high and low volatility, where periods of high volatility are often followed by similar periods.

The GARCH model consists of two main components:

1. Autoregressive Component (ARCH): The ARCH component models the past squared returns, which represent the past volatility, using an autoregressive process. It assumes that the past volatility has a relationship with the current volatility.
2. Moving Average Component (GARCH): The GARCH component models the past conditional variance (squared errors) using a moving average process. It enables for the persistence of past volatility shocks to influence the current conditional variance.

The general GARCH (p, q) model is represented as:

$$\sigma_t^2 = \omega + \sum_{i=1}^{p} \alpha_i \epsilon_{t-i}^2 + \sum_{j=1}^{q} \beta_j \sigma_{t-j}^2$$

Equation 2.12: GARCH model formula.

where:

σ_t^2 is the conditional variance or volatility at time t.

ω is the constant term.

α_i and β_j are parameters to be estimated.

ϵ_{t-i}^2 are the squared residuals or squared errors at lag i.

The GARCH model is widely used in finance for volatility forecasting, risk management, and option pricing. By capturing volatility clustering, it provides valuable insights into the dynamics of financial markets and helps in making informed investment decisions.

2.7.2 ARIMA for Time Series Modelling

The Autoregressive Integrated Moving Average (ARIMA) model is a time series forecasting model commonly used in statistics and econometrics to analyse and predict future values of a univariate time series. It was developed by George Box and Gwilym Jenkins in the 1970s and has become one of the most widely used models for time series analysis [6], [24].

ARIMA models combine three main components to capture the underlying patterns in the data:

1. Autoregressive (AR) Component: This component models the relationship between the current observation and its previous values. It assumes that the current value of the time series can be explained by a linear combination of its lagged values.
2. Integrated (I) Component: The integrated component is used to make the time series stationary by differencing the data. Stationarity is an important assumption in time series analysis, and differencing helps to stabilize the mean and variance of the series.
3. Moving Average (MA) Component: This component models the relationship between the current observation and the errors (or residuals) from previous observations. It assumes that the current value of the time series can be explained by a linear combination of its past errors.

The general notation for an ARIMA (p, d, q) model is:

$$Y_t = \mathrm{c} + \sum_{i=1}^{p} \emptyset_i Y_{t-i} + \sum_{j=1}^{q} \theta_j \varepsilon_{t-j} + \varepsilon_t$$

Equation 2.13: ARIMA model formula.

where:

Y_t is the value of the time series at time t.

c is a constant term.

$\emptyset_i$ and θ_j are parameters to be estimated.

p is the order of the autoregressive component (AR).

d is the degree of differencing required to make the time series stationary (integrated component).

q is the order of the moving average component (MA).

ε_t represents the error term, which is assumed to be white noise.

ARIMA models are widely used for time series forecasting in various fields, including economics, finance, engineering, and environmental sciences. They are especially useful when dealing with non-stationary time series data, where the values may exhibit trends or seasonality. ARIMA models can provide valuable insights into the underlying patterns in the data and help make accurate predictions for future values.

2.8 Conclusion

Successfully navigating the stock market requires a wealth of experience, constant monitoring, and rigorous testing. Investors continuously develop models to simulate various market conditions, ensuring they remain well-prepared for any scenario. Accurate predictions of stock market metrics, including daily returns, trading volumes, and key market indicators such as volatility, offer significant advantages to investors and portfolio managers, placing them on the winning side of trades.

The novel prediction methods studied in this research hold the potential to enhance existing pricing and trading models. Moreover, they can serve as valuable tools for risk management, helping investors maximize profits and minimize losses, the ultimate goal for all market participants. By leveraging these innovative approaches, investors can gain a competitive edge and optimize their decision-making process in the dynamic world of finance.

Chapter 3

Deep Learning Networks, Loss Function, and Multitasking

Chapter 3 undertakes and in-depth analysis of deep learning networks, with a particular focus on recurrent neural networks (RNNs), long short-term memory (LSTM) networks, and gated recurrent units (GRUs). These specialized architectures have gained prominence due to their exceptional performance in handling sequential data, making them highly relevant for analysing financial time series data.

The chapter also analyses the realm of multi-task learning and its intricate connection with loss functions. Multi-Task learning, a dynamic paradigm in machine learning, takes on the ambitious challenge of training a single model to effectively handle multiple interrelated tasks concurrently. The primary aim is to leverage shared insights and representations across these tasks to elevate the overall performance of the model.

Loss function forms the cornerstone of the examination, marries multi-task learning with loss functions. The chapter dissects the complex interplay between these two components, shedding light on how task-specific loss functions are seamlessly integrated into the multi-task learning framework. This integration is carefully orchestrated to strike a balance—ensuring the optimization of each task performance while harnessing the synergistic power of shared knowledge across tasks.

The relationship between gradient descent and loss functions is fundamental in the optimization process of machine learning models. Gradient descent serves as the guiding force that minimizes the chosen loss function, shaping the model parameter values to enhance predictive accuracy.

Multi-Task learning unveils the fundamental principles that underlie this approach, showcasing its capacity to enhance both efficiency and effectiveness, particularly in scenarios where tasks demonstrate inherent correlations or interdependencies. This method revolves around the joint optimization of a model parameters across various tasks, with the overarching goal of achieving superior generalization, heightened accuracy, and fortified robustness. This marks a notable departure from the conventional practice of training distinct models for each individual task.

The chapter covers the following key topics:

LSTM Deep Learning Networks: This section conducts a study on the fundamentals of RNNs, examining how they process sequential data by maintaining hidden states that capture past information. It explains how RNNs suffer from the vanishing gradient problem and introduces the need for more advanced architectures.

The section is more focused on LSTM networks, a type of RNN designed to overcome the vanishing gradient problem. It explains the LSTM architecture, which incorporates memory cells and gates to selectively retain and forget information. LSTM ability to handle long-term dependencies makes it particularly useful for financial time series data.

GRU Deep Learning Networks: This section introduces GRU networks, another variant of RNNs that address the vanishing gradient problem. GRUs use fewer parameters than LSTM, making them computationally more effective while still maintaining effective performance in modelling sequential data.

Loss Function: This section delineates the examination of various functions applicable to timeseries and financial market data, encompassing domains such as stock market prices.

Gradient Descent: This section provides a high-level elucidation of its mechanics and elucidates its interconnectedness with loss functions within the machine learning realm.

Multi-Task Learning: The research investigation then extends to the realm of multi-task learning. This section examines into the advantages reaped from embracing multi-task learning, while simultaneously delving into its implications for loss functions and the holistic loss incurred by a composite model composed of diverse tasks.

Training and Optimization: This section discusses the training process for deep learning networks, including techniques for optimizing model parameters and avoiding overfitting. It may cover topics such as gradient descent, backpropagation through time, and hyperparameter tuning.

Limitations and Future Directions: This section outlines the limitations and challenges of using RNN, LSTM, and GRU networks in financial applications. It also discusses potential future developments and research directions to improve the performance and applicability of these models.

3.1 LSTM Deep Learning Network

Recurrent Neural Networks (RNN) were introduced in 1986 by Rumelhart [25] to resolve gradient errors associated with one of the simplest, commonly deployed form of neural network, Feed Forward Neural Networks (FFNN).

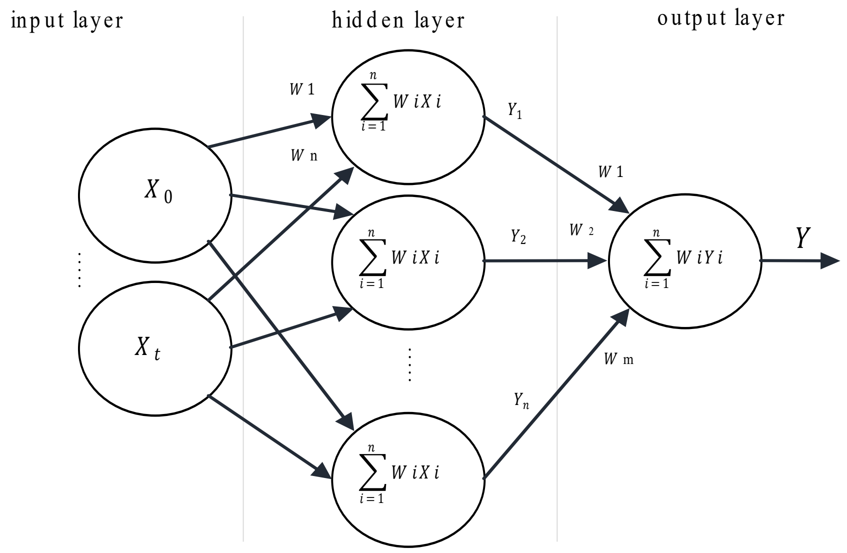


Figure 3.1 Feed Forward Neural Networks (FFNN).

The architecture of Recurrent Neural Networks (RNNs) grants each cell within the network the capability to process input data and subsequently transmit the outcomes to the subsequent node. This intrinsic feature empowers RNNs to emerge as one of the most proficient machine learning algorithms for handling sequential data. You can observe the standard network and cell structures of an RNN in Figure 4.1 and Figure 4.2, respectively, which illustrate this sequential data processing capability.

Long term dependencies have always been a significant weakness of ordinary RNN, the network struggles to handle gaps between relevant information when it becomes large, in other words, ordinary large RNN suffer from short memory issues, this phenomenon is referred to as Vanishing Gradient Problem [26].

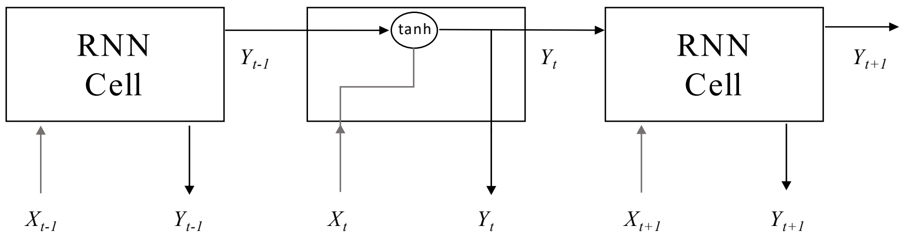


Figure 3.2: RNN network architecture.

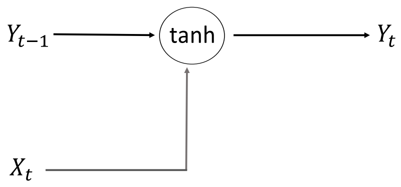


Figure 3.3: RNN cell architecture.

$$Y_t = tanh(W[Y_{t-1}, X_t])$$

Equation 3.1: RNN cell formula.

Where:

X_t is the input.

Y_{t-1} in the output from previous RNN block.

Y_t is the output of the current RNN block.

This limitation can cause several difficulties to the algorithms accuracy which was the rationale behind looking for a new improved version of RNN.

The solution to vanishing gradient problem came in 1997 by Hochreiter & Schmidhuber [27] when they introduced the new Long Short-Term Memory (LSTM) algorithm which was capable of learning long term dependencies and yet, appreciate all the great features of sequential processing capabilities inherited from the parent RNN architecture. A typical LSTM network is illustrated in Figure 4.4.

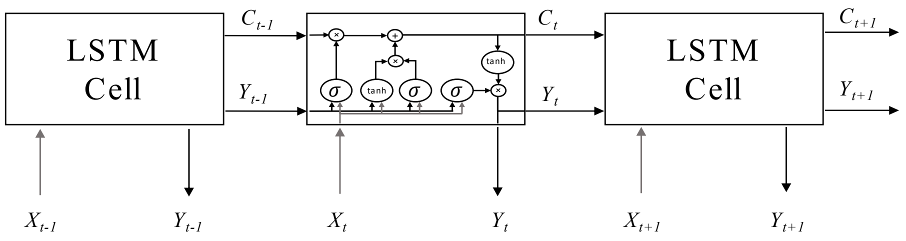


Figure 3.4: LSTM network architecture.

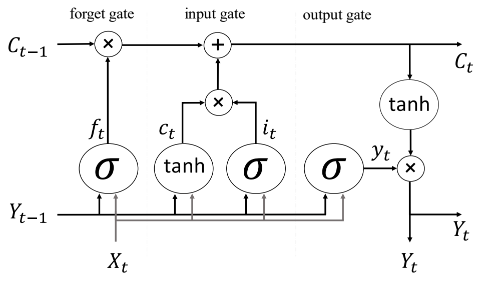


Figure 3.5: LSTM cell architecture.

Within LSTM, a pivotal element known as the cell state, denoted as Ct, plays a central role. It is visually represented as a horizontal line spanning across the network top.

This cell state is meticulously managed by gates, which are responsible for executing multiplication and summation operations based on the sigmoid output. These operations serve as the gatekeepers, determining which pieces of information to retain and which to discard. This inherent mechanism endows LSTM with a remarkable capability for effectively predicting sequential data by selectively retaining and processing relevant information.

The internal structure of a single LSTM block is illustrated in Figure 4.5.

$$f_t = \sigma\left(W_f[Y_{t-1}, X_t] + b_f\right)$$

$$i_t = \sigma(W_i[Y_{t-1}, X_t] + b_i)$$

$$y_t = \sigma\left(W_y[Y_{t-1}, X_t] + b_y\right)$$

$$c_t = tanh(W_c[Y_{t-1}, X_t])$$

$$C_t = (f_t \times C_{t-1}) + (i_t \times c_t)$$

$$Y_t = y_t \times \tanh(C_t)$$

Equation 3.2: LSTM cell formulas.

Where:

f_t is the forget function, W_f forget gate weight, and b_f is forget bias.

i_t is the input function, W_i input gate weight, and b_i is input bias.

y_t is the output function, W_y output gate weight, and b_y is output bias.

X_t is the input.

C_t cell state.

Y_{t-1} in the output from previous RNN block.

Y_t is the output of the current RNN block.

In LSTM network, there are three main gates that control the flow of information and states within the LSTM cell: the input gate, the forget gate, and the output gate. These gates help the LSTM cell to selectively process and store information over different time steps, enabling the network to capture long-range dependencies and mitigate the vanishing gradient problem [27], [28].

Forget Gate (f_t): The forget gate determines what information from the previous cell state C_{t-1} should be retained or discarded. It takes a combination of the current input X_t and the previous hidden state Y_{t-1} as input and passes it through a sigmoid activation function. The output of the forget gate f_t is then multiplied elementwise with the previous cell state to decide which parts of the cell state should be forgotten.

Input Gate (i_t): The input gate determines how much new information should be added to the current cell state C_t takes a combination of the current input X_t and the previous hidden state Y_{t-1} as input and passes it through a sigmoid activation function. This gate decides which values from the input should be updated into the cell state. Additionally, a tanh activation function processes the same input to generate a candidate cell state update C_t. The input gate then modulates the candidate update using the sigmoid-activated input gate output, controlling which parts of the candidate update should be added to the cell state.

Output Gate (y_t): The output gate decides what information from the current cell state C_t should be used to produce the output Y_t of the LSTM cell at the current time step. Such as the input and forget gates, it takes a combination of the current input X_t and the previous hidden state Y_{t-1} as input and passes it through a sigmoid activation function. The current cell state X_t is then processed by the tanh activation function to bring it to the desired output range. The output gate sigmoid-activated output y_t then modulates the tanh-activated cell state to generate the final output of the LSTM cell.

By using these gates, an LSTM cell can effectively control the flow of information, selectively update the cell state, and produce meaningful outputs at each time step, making it well-suited for tasks involving sequential and temporal data.

Each gate is also controlled by weights and bias, this is the key feature in LSTM as the weights are continuously updated to reflect gradient and errors from data way back. Without this feature, the RNN will be biased to recent data changes only, hence, accuracy will decline as processing continue.

In LSTM network, a bias parameter is a learnable scalar added to the weighted sum of inputs and recurrent states before passing through activation functions. Each LSTM cell has three different bias terms: input bias, forget bias, and output bias [27], [28].

Input Bias (b_i): This bias term is added to the linear combination of the input data and the previous hidden state before passing through the sigmoid activation function. It controls the amount of new information that should be added to the cell state. A higher input bias can encourage the LSTM cell to consider more of the current input.

Forget Bias (b_f): The forget bias is added to the linear combination of the input data and the previous hidden state before passing through the sigmoid activation function that computes the forget gate. This bias helps control the extent to which the cell should forget the previous cell state. A higher forget bias can lead the LSTM cell to forget more of the previous cell state.

Output Bias (b_i): The output bias is added to the linear combination of the input data and the previous hidden state before passing through the sigmoid activation function that computes the output gate. This bias influences the amount of information that should be used from the cell state to produce the output of the LSTM cell.

These bias terms are crucial for the functioning of the LSTM cell because they enable the network to adjust the balance between input, forget, and output behaviours. Bias parameters are learned during the training process along with the other weights of the network, enabling the LSTM to adapt to the specific characteristics of the data and the task.

Bias terms in LSTMs play a role in ensuring the model ability to capture and retain important information over time, making them a key component of the LSTM architecture.

In LSTM and deep learning in general, both the hyperbolic tangent (tanh) and sigmoid activation functions are used, but they serve different purposes based on their characteristics and properties.

Tanh (Hyperbolic Tangent) Function

The tanh function squashes input values to the range of [-1, 1]. It is zero-cantered, meaning that its outputs have a mean of approximately zero. Tanh is often used in the hidden layers of neural networks, particularly in recurrent neural networks (RNNs) and long short-term memory (LSTM) networks, to introduce non-linearity while enabling the model to capture more complex patterns in the data. Tanh can help prevent the "vanishing gradient" problem compared to sigmoid because it has a larger output range, which can result in faster convergence during training. Figure 4.2 illustrates a tanh function output [11], [12].

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Equation 3.3. Tanh activation function formula.

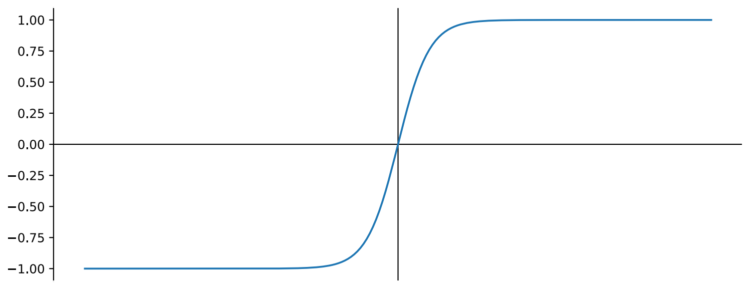


Figure 3.6: Tanh activation function output.

Sigmoid Function

The sigmoid function maps input values to the range of [0, 1]. It is commonly used in binary classification problems, where it transforms the output into a probability value between 0 and 1. The sigmoid function is also used in the output layer of networks for tasks such as binary classification, as it conveniently converts the network final output into a probability score [11], [12].

$$S(x) = \frac{1}{1 + e^{x}}$$

Equation 3.4: Sigmoid activation function formula.

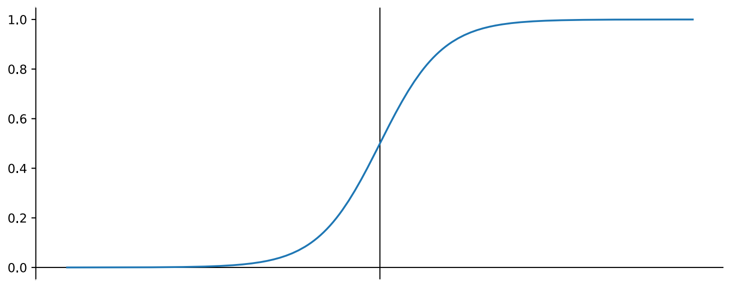


Figure 3.7: Sigmoid activation function output.

tanh is often preferred in hidden layers of networks where the zero-centred property can help with optimization, and it can mitigate the vanishing gradient issue. Sigmoid is suitable for tasks where you need binary classification probabilities or when you want to restrict outputs to a specific range. However, in many cases, modern deep learning architectures, such as Rectified Linear Unit (ReLU) and its variants, have become more popular choices for activation functions due to their improved training properties and performance.

3.2 GRU Deep Learning Network

Gated Recurrent Unit (GRU) algorithm has been introduced recently by Chao et al [29], it simplified the original LSTM by combining the input and forget into a single gate and merged cell and hidden state.

GRU boasts a simpler architecture when compared to LSTM, making it more straightforward to implement and quicker to train. Moreover, it features fewer parameters, which becomes advantageous when handling limited data or working with computational resource constraints.

Both LSTM and GRU leverage gating mechanisms to regulate information flow, but GRU takes it a step further by combining the forget and input gates into a single update gate, resulting in a more streamlined structure.

In scenarios where long sequences of data are involved, GRU tends to outperform LSTM, as it has demonstrated improved resistance to the vanishing gradient problem. Additionally, thanks to its simplified architecture and reduced parameter count, GRU exhibits greater computational efficiency compared to LSTM, rendering it a preferred option for specific applications.

Figure 4.6 illustrates the structure of a GRU network, where X_t indicates the input and Y_t indicates the output.

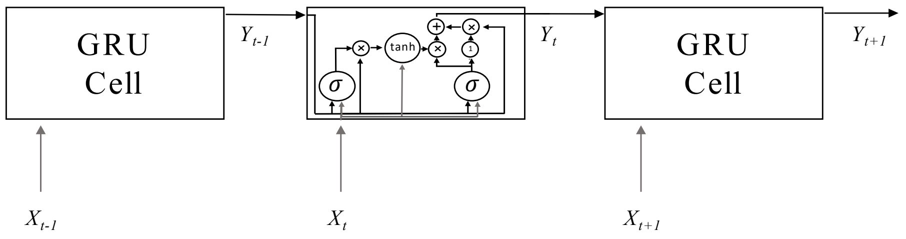


Figure 3.8: GRU network architecture.

$$r_t = \sigma(W_r[Y_{t-1}, X_t] + b_r)$$

$$u_t = \sigma(W_u[Y_{t-1}, X_t] + b_u)$$

$$y_t = \tanh(W_y[(r_t \times Y_{t-1}), X_t] + b_y)$$

$$Y_t = \big((1 - u_t) \times Y_{t-1}\big) + (u_t \times y_t)$$

Equation 3.5: GRU cell formulas.

Where:

r_t is the reset function, W_r reset gate weight, and b_r is reset bias.

u_t is the update function, W_u update gate weight, and b_u is update bias.

y_t candidate output state, W_y candidate state weight, and b_y is candidate state bias.

X_t is the input.

Y_{t-1} in the output from previous RNN block.

Y_t is the output of the current RNN block.

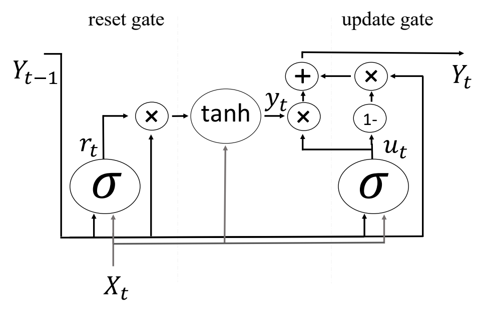


Figure 3.9: GRU cell architecture.

In GRU neural network architecture, there are two main gates that regulate the flow of information and control the memory within the GRU cell: the update gate and the reset gate. These gates help the GRU to selectively process and update its internal state, enabling it to capture temporal dependencies and patterns in sequential data [11], [29].

Update Gate (u_t): The update gate determines how much of the previous internal state Y_{t-1} should be retained and how much of the current input X_t should be incorporated into the new internal state Y_t. It takes the concatenation of the current input and the previous hidden state as input and passes it through a sigmoid activation function. The output of the update gate controls the trade-off between retaining the past state and incorporating new information.

Reset Gate (r_t): The reset gate determines how much of the previous hidden state Y_{t-1} should be forgotten when calculating the new candidate state y_t. Similarly to the update gate, it takes the concatenation of the current input and the previous hidden state as input and passes it through a sigmoid activation function. The reset gate output controls the extent to which the past hidden state is reset when computing the candidate update.

In a GRU neural network, bias terms are additional parameters that influence the behaviour of the gates and the candidate state. There are typically bias terms associated with the update gate, the reset gate, and the candidate state computation [11], [29].

Update Gate Bias (b_u): The update gate bias is added to the linear combination of the current input X_t and the previous hidden state Y_{t-1} before passing through the sigmoid activation function that computes the update gate. This bias can influence how much the model prefers to update the internal state based on the input and the previous state.

Reset Gate Bias (b_r): The reset gate bias is added to the linear combination of the current input X_t and the previous hidden state Y_{t-1} before passing through the sigmoid activation function that computes the reset gate. This bias can control the tendency of the reset gate to reset or forget the previous hidden state.

Candidate State Bias (b_y): The candidate state bias is added to the linear combination of the current input X_t and the previous hidden state Y_{t-1} before passing through the tanh activation function that computes the candidate state y_t. This bias can influence the initial behaviour of the candidate state computation.

The inclusion of bias terms within the GRU cell is a dynamic process integrated into the training phase, wherein these bias terms are learned alongside the other parameters of the GRU cell. This learning process is vital as it enables the model to exhibit adaptability and precision by fine-tuning the operations of the gates and the candidate state. These bias terms essentially serve as adjustable parameters that facilitate the alignment of the model behaviour with the unique characteristics of the data being processed, thereby enhancing the model performance in executing specific tasks. This fine-tuning capability contributes significantly to the model effectiveness in addressing a diverse range of data and tasks.

3.3 Loss Function

A loss function, also known as a cost function or objective function, is a crucial component in machine learning algorithms, particularly during the training phase. It quantifies the difference between the predicted values of the model and the actual target values in the dataset. The goal of training is to minimize this loss function, which essentially measures how well the model predictions match the ground truth.

In the context of supervised learning, where the algorithm learns from labelled data, the loss function serves as a guide for adjusting the model parameters to improve its performance. Different types of machine learning tasks (such as classification, regression, or even more specialized tasks) often require specific types of loss functions.

Below, presents several instances of loss functions applicable to machine learning tasks involving timeseries and financial data.

Mean Squared Error (MSE) is used in regression tasks, MSE calculates the average squared difference between the predicted and actual target values. It penalizes larger errors more heavily [11], [12].

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y_i})^2$$

Equation 3.6: Mean Squared Error (MSE) formula.

Where:

y_i is the actual target value.

$\hat{y_i}$ is the predicted value.

n is the number of data points used.

Root Mean Squared Error (RMSE) is a variation of MSE that further takes the square root of the average squared differences [11]. It provides a more interpretable measure in the same unit as the original values. RMSE is expressed is expressed as:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y_i})^2}$$

Equation 3.7: Root Mean Squared Error (RMSE) formula.

RMSE imposes a greater penalty on larger prediction errors compared to MSE due to the square root operation, which accentuates the influence of these errors. RMSE finds frequent use when the intention is to gauge prediction error in the original unit of the target variable. Furthermore, RMSE offers the advantage of rendering a measure in the same unit as the original data, facilitating a more intuitive interpretation in real-world contexts.

Conversely, MSE is expressed in squared units and is often favoured when there is substantial variability in the magnitudes of predicted and actual values. This preference arises as MSE aids in mitigating the sway of outliers that may skew the assessment.

It is imperative to note that RMSE propensity to magnify the impact of large errors stems from the square root operation. This characteristic can be particularly pertinent when there is a desire to underscore the significance of substantial deviations from the predicted values.

Huber loss is an alternative to MSE for regression tasks, Huber loss is less sensitive to outliers and can provide a balance between the robustness of MSE and the absolute error loss (MAE) [30].

$$f(x) = \begin{cases} \frac{1}{2}(y - y^{\wedge})^2, & if\ |y - y^{\wedge}| \leq \delta \\ \delta(|y - y^{\wedge}| - \frac{\delta}{2}), & otherwise \end{cases}$$

Equation 3.8: Hyper Loss formula.

Where:

δ is a threshold parameter.

3.4 Gradient Descent

Gradient descent is a fundamental optimization algorithm widely deployed in machine learning and deep learning to iteratively minimize a loss function and enhance the performance of a model. It operates by iteratively adjusting the model parameters in the direction that reduces the value of the loss function, thereby reaching a point of optimal parameter values [11], [12].

The algorithm core idea is rooted in calculus and the concept of gradients. The gradient of a function at a specific point indicates the direction of the steepest increase of the function. In the context of gradient descent, the negative gradient points in the direction of the steepest decrease, which is the direction in which the algorithm seeks to move to minimize the loss.

Gradient descent algorithm typically works in the following order:

Step 1: Initialization: Initialize the model parameters with arbitrary values.

Step 2: Compute Gradient: Calculate the gradient of the loss function with respect to the model parameters. This involves computing partial derivatives for each parameter.

Step 3: Update Parameters: Adjust the parameters in the opposite direction of the gradient by a certain step size, often referred to as the learning rate. The learning rate determines the size of the steps taken in each iteration.

Step 4: Repeat steps 2 and 3 iteratively until convergence or a predetermined number of iterations is reached.

For a deeper understanding, the steps above can be elucidated in the following manner:

The gradient decent start by fitting a hypothesis function $h_\theta(x)$ as shown in Equation 3.4 [11], [12] which is represents step (1).

$$h_\theta(x) = \theta_0 + \theta_1 x$$

Equation 3.9: Gradient descent – Hypothesis formula.

Where:

$h_\theta(x)$ is the hypothesis function.

$\theta_0$is the first parameter value.

θ_1 is the next parameter value.

x is the input value.

The cost function of the gradient decent is the squared difference between the hypothesis and the actual value. This function is generally referred to as loss function.

Calculation by this formula [11], [12] represent step (2).

$$J(\theta) = \frac{1}{2m}\sum_{i}^{m}\left[h_\theta(x^{(i)}) - y^{(i)}\right]^2$$

Equation 3.10: Gradient descent – Cost function formula.

Where:

$J(\theta)$ is the cost function.

$h_\theta(x^{(i)})$ is the predicted value.

$y^{(i)}$ is the actual value.

The new parameter θ_{new} is calculated using the learning rate (α). This calculation represents step (3).

$$\theta_{new} = \theta_{old} - (\alpha \times \nabla J(\theta_{old}))$$

Equation 3.11: Gradient descent formula.

Where:

θ_{new} represents the updated parameter values.

θ_{old} represents the current parameter values.

α is the learning rate, determining the step size for each iteration.

$\nabla J(\theta_{old})$ is the gradient of the loss function J with respect to the parameters θ_{old}.

Figure 3.1 depicts the sequence of actions undertaken by gradient descent to locate the minimum data point, considering both modest and substantial learning rates [31].

Opting for a reduced learning rate, while enhancing precision, comes at the cost of extended computational time needed to reach the minimum point. Conversely, embracing a higher learning rate enables for more substantial steps during training, yet it introduces the potential risk of overshooting the minimum point. This trade-off between precision and computational efficiency is a key consideration in the optimization process.

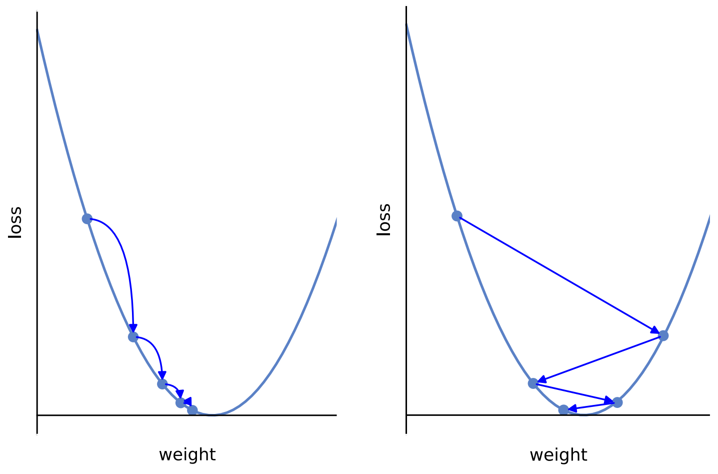


Figure 3.10: Gradient descent - small(left) and large(right) learning rates.

There are different variants of gradient descent, Batch gradient descent, Stochastic gradient descent, and Hybrid gradient descent [31], [32].

Batch gradient descent computes the gradient using the entire training dataset at each iteration. It provides a more accurate estimate of the gradient but can be computationally expensive for large datasets.

Stochastic gradient descent computes the gradient using only a single training sample at each iteration. It is computationally more effective but can have more noisy updates.

Hybrid gradient descent, aften referred to as "Mini-batch gradient decent", computes the gradient using a small subset of the training data. This combines the benefits of both batch and stochastic gradient descent.

In deep learning, gradient descent is a fundamental optimization algorithm used to update the parameters of a neural network in order to minimize the loss function. While gradient descent is powerful and widely used, it can encounter several issues when applied to deep learning models.

Vanishing gradients, Exploding Gradients, local minimum, and saddle point are known challenges to recurrent deep neural networks during model training with gradient descent and backpropagation.

Vanishing gradients scenario arises when gradients become exceedingly small. While backpropagating through the network layers, the gradient consistently decreases, leading to gradual learning in earlier layers compared to later ones. This phenomenon may culminate in weight parameter updates dwindling to insignificance, effectively rendering the algorithm dormant [31], [33].

The exploding gradients situation emerges when gradients attain excessive magnitudes, inducing model instability. Consequently, the model weights grow disproportionately large, ultimately diverging to NaN (Not-a-Number) representations. To address this, a potential remedy involves harnessing dimensionality reduction techniques, which mitigate model complexity and promote stability [31], [34].

A local minimum is a point in the parameter space where the loss function has the lowest value within a certain neighbourhood of that point. In other words, the loss function is lower at this point than at nearby points. However, it is important to note that a local minimum might not be the global minimum (lowest value of the entire loss function), which is the ultimate goal in optimization. Gradient descent can sometimes get stuck in a local minimum, preventing it from finding the global minimum [35].

A saddle point is a point in the parameter space where the loss function has a flat region surrounded by higher values in some directions and lower values in other directions. At a saddle point, the gradient of the loss function is close to zero, which makes it difficult for gradient descent to determine the optimal direction to move in. Saddle points can slow down the convergence of gradient descent as it may linger in the flat region without making significant progress toward the minimum [31], [32], [36].

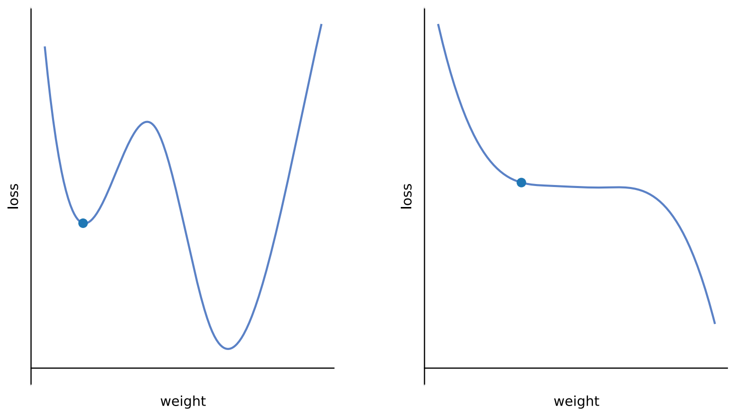


Figure 3.11: Gradient descent - local minimum(left) and saddle point(right).

The concept of a loss function is tightly intertwined with the optimization process in machine learning, particularly gradient descent. A loss function quantifies the disparity between a model predictions and the actual target values, serving as a crucial guide for model refinement.

Gradient descent, on the other hand, is an optimization algorithm aimed at minimizing the loss function by iteratively adjusting the model parameters. It operates by computing the gradient of the loss function with respect to the model parameters. This gradient points in the direction of steepest ascent, so to minimize the loss, the algorithm takes steps in the opposite direction (downhill) by subtracting a fraction of the gradient (learning rate) from the current parameters. This process continues iteratively until convergence, or a predefined stopping criterion is met.

3.5 Multi-Task Learning

Multi-Task learning stands as a machine learning paradigm wherein a model is trained to adeptly handle multiple interconnected tasks concurrently. This method capitalizes on shared insights and representations across these tasks to heighten overall performance. Within the realm of multi-task learning, the model aspires to enhance its proficiency in each task by gleaning insights from the information embedded within other tasks [37].

This approach proves particularly advantageous when tasks exhibit common underlying structures or relationships, empowering the model to glean knowledge more effectively and effectively. Through the collective optimization of model parameters across multiple tasks, multi-task learning yields improved generalization, heightened accuracy, and augmented robustness in comparison to the approach of training separate models for each task.

Multi-Task learning finds particular utility in situations where labelled data is scarce for individual tasks, but ample for the amalgamated task set. Moreover, it aids in refining the acquisition of representations that encapsulate pivotal features across tasks, consequently elevating the efficacy of feature extraction.

Instances of multi-task learning applications encompass the domain of natural language processing, encompassing tasks such as part-of-speech tagging, named entity recognition, and syntactic parsing—tasks interwoven by linguistic dependencies.

Within the domain of computer vision, the utilization of multi-task learning extends to pursuits encompassing objectives such as object detection, image segmentation, and facial recognition.

Below are the main benefits of Multitasking [37], [38], [39], [40] :

Efficiency: Multitasking enables the effective utilization of computational resources by training a single model instead of maintaining multiple models for each task. This consolidation can lead to reduced memory and processing requirements.

Shared Representations: Multi-Task learning encourages the model to discover shared representations that capture the essential features across tasks. This shared knowledge can lead to better generalization on unseen data and improved adaptability to changes in the input distribution.

Data Efficiency: In scenarios with limited data, multitasking can be particularly advantageous. The model can leverage information from related tasks to improve performance on tasks with sparse data.

Regularization: Multitasking can act as a form of regularization, preventing overfitting by encouraging the model to focus on the most important features shared across tasks.

In the context of multi-task learning, where a single model is trained to perform multiple tasks simultaneously, the concept of loss functions becomes even more significant. Each task in multi-task learning typically has its own associated loss function, reflecting the specific objective of that task. The challenge lies in effectively combining these task-specific loss functions to train the model.

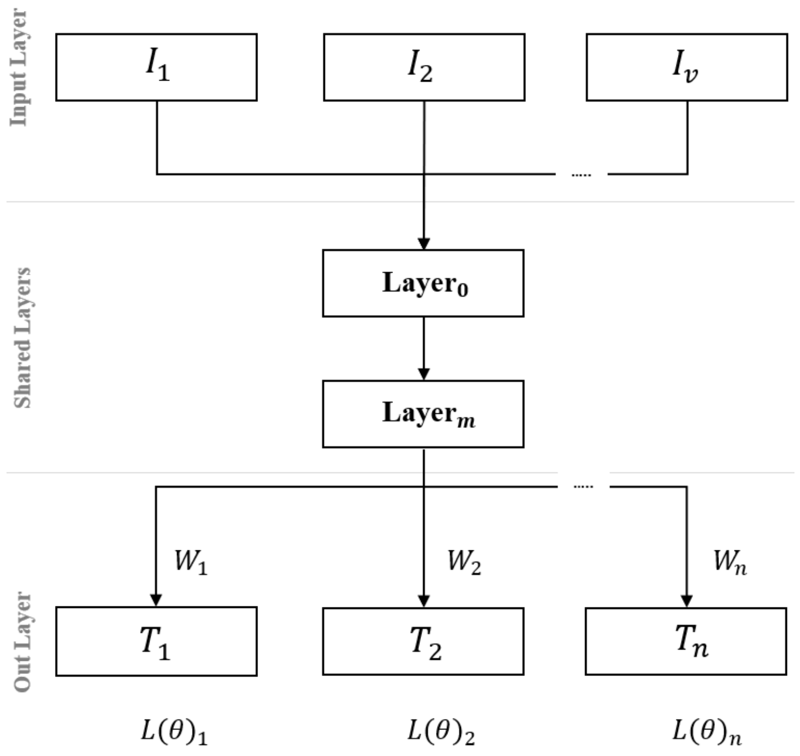


Figure 3.12: Multitasking architecture.

The overarching goal in multi-task learning is to find a balance between optimizing each task performance while leveraging shared knowledge across tasks. The formulation of the combined loss function depends on how the tasks are related and how the model parameters are shared among them.

In multi-task learning, the individual task loss functions are combined into a joint objective function that the model aims to minimize. This joint objective function can take different forms, depending on how the tasks are weighted relative to each other. It may include summing the task-specific loss functions with appropriate weights:

$$Loss_{Total} = \sum_{i}^{n} W_i \times L_i(\theta)$$

Equation 3.12. Model Loss function formula.

Where:

W_i represents the weight.

$L_i(\theta)$ represents the loss function for task T_i

To encourage shared knowledge and prevent overfitting, regularization terms can be added to the joint objective function. These terms may penalize the model for diverging too much from shared parameters.

The choice of W_i values can reflect the relative importance of each task. Tasks with more available data or higher priority may receive higher weights to guide the model learning.

In some cases, dynamic weighting or adaptive approaches can be used to adjust the importance of each task during training, based on factors such as task difficulty or the model current performance on each task.

Loss scaling can be applied to ensure that the gradients contributed by each task are balanced, preventing one task from dominating the optimization process.

Multi-Task learning success relies on finding an equilibrium between enhancing individual task performance and leveraging shared information. The loss functions play a pivotal role in achieving this balance, guiding the model to learn useful representations that benefit all tasks simultaneously.

3.6 Training and Optimization

Training and optimization of LSTM (Long Short-Term Memory) and GRU (Gated Recurrent Unit) networks include several key steps to ensure their effectiveness in learning from financial time series data. Detailed overview of the steps is listed below [11], [26], [27], [29].

Data Preprocessing

Before training the LSTM or GRU model, the financial time series data needs to be pre-processed. This includes steps such as handling missing values, normalization or scaling of the data, and splitting it into training, validation, and test sets.

Model Architecture

The architecture of the LSTM or GRU network is defined, specifying the number of layers, the number of memory cells (units) in each layer, and other hyperparameters. The choice of architecture depends on the complexity of the data and the prediction task.

Loss Function

For training a regression model, a common choice for the loss function is Mean Squared Error (MSE), which measures the difference between the predicted values and the actual targets. For classification tasks, categorical cross-entropy or binary cross-entropy is used.

Optimization Algorithm

The optimization algorithm is responsible for updating the model parameters (weights and biases) during training to minimize the loss function. Popular optimization algorithms for LSTM and GRU networks include Adam, RMSprop, and SGD (Stochastic Gradient Descent).

Root Mean Square Propagation (RMSProp) adjusts the learning rate based on the magnitudes of recent gradients. It is particularly useful for handling sparse data and can be beneficial when dealing with time series that exhibit varying levels of volatility.

Adaptive Moment Estimation (Adam) is an adaptive optimization algorithm that combines elements of both momentum and RMSProp. It adjusts the learning rate for each parameter individually, making it suitable for non-stationary time series data where the data distribution might change over time.

Stochastic Gradient Descent (SGD) is a widely used optimization algorithm that updates model parameters based on the gradient of the loss function with respect to a small subset of the training data. It is beneficial for large datasets and can be adapted for time series data by considering sequences of data points as mini batches.

Learning Rate

The learning rate is a hyperparameter that controls the step size at which the optimization algorithm updates the model parameters. A suitable learning rate is essential for faster convergence and avoiding overshooting the optimal solution.

Backpropagation Through Time (BPTT)

LSTM and GRU networks are recurrent architectures, so training involves propagating the error gradient backward through time. BPTT helps the model learn from previous time steps and update the memory cells accordingly.

Regularization

Regularization techniques such as dropout and L2 regularization are applied to prevent overfitting. Dropout randomly deactivates neurons during training, and L2 regularization adds a penalty to the loss function for large weights, both of which improve generalization.

Hyperparameter Tuning

The hyperparameters, such as the number of layers, the number of units in each layer, learning rate, and dropout rate, are fine-tuned to optimize the model performance. This is typically done using techniques such as grid search or random search.

Early Stopping

To avoid overfitting, early stopping is deployed. It monitors the performance on a validation set during training and stops the training process when the performance starts to degrade.

Evaluation

After training, the LSTM or GRU model is evaluated on the test set to assess its performance on unseen data. Metrics such as Mean Absolute Error (MAE) or Mean Squared Error (MSE) are used to measure the model accuracy.

By carefully selecting the model architecture, loss function, optimization algorithm, and hyperparameters, and using regularization techniques, the LSTM and GRU networks can be effectively trained and optimized to make accurate predictions for financial time series data, enabling optimised portfolio management, risk management, and decision-making in financial applications.

3.7 Limitations and Future Directions

Deep learning has revolutionized various fields, including natural language processing, time series analysis, and more. LSTM (Long Short-Term Memory) and GRU (Gated Recurrent Unit) networks are two popular types of recurrent neural networks (RNNs) that have gained significant attention for their ability to model sequential data. This research examines the strengths and weaknesses of LSTM and GRU networks, shedding light on their computational complexity, susceptibility to overfitting, training time, gradient-related challenges, memory requirements, interpretability, and the cold start problem.

Computational Complexity and Training Time

LSTM and GRU networks offer sophisticated memory mechanisms to capture long-term dependencies in sequential data, but these advantages come at a cost. Both networks have higher computational complexity compared to simpler feedforward neural networks. This complexity slows down the training process, making them slower to train and demanding more memory for storage. In particular, the backpropagation through time process used in training RNNs can be time-consuming, especially for large datasets or deep architectures [11], [12].

Overfitting

One of the challenges faced by LSTM and GRU networks is the risk of overfitting, particularly when dealing with small datasets. Overfitting occurs when the model becomes too specific to the training data and fails to generalize well to unseen data. Regularization techniques such as dropout and L2 regularization are commonly used to mitigate this issue and improve the model ability to generalize [11], [12].

Vanishing and Exploding Gradients

LSTM and GRU networks may encounter the vanishing and exploding gradient problems during the training process. The vanishing gradient problem occurs when gradients diminish exponentially during backpropagation, making it challenging to learn long-range dependencies in the data. On the other hand, the exploding gradient problem results in gradients becoming too large, causing unstable training. This topic is discussed in detail in section 3.2 [11], [12], [31], [33].

Parameter Sensitivity

The performance of LSTM and GRU networks is highly sensitive to hyperparameter tuning. Careful adjustment of hyperparameters is essential to achieve optimal performance, as small changes can significantly impact the model results.

Memory Requirements

The memory mechanisms in LSTM and GRU networks enable them to store information from past time steps, facilitating their ability to learn from sequences. However, this advantage comes with a trade-off as they often require substantial memory, which can be a limitation when working with large sequences or resource-constrained devices.

Interpretability

LSTM and GRU networks are considered black-box models, meaning their internal workings can be challenging to interpret. Understanding how these models arrive at their predictions can be crucial in applications that require transparency and details of implementation or methods used.

Cold Start Problem

LSTM and GRU networks may face difficulties in making accurate predictions during the initial time steps, especially when there is limited historical data available. This "cold start problem" can be a limitation in certain time series forecasting scenarios.

3.8 Conclusions

LSTM and GRU networks are highly effective and powerful architectures for sequential data processing tasks. They have proven to be very successful in a wide range of applications, including natural language processing, speech recognition, time series forecasting, sentiment analysis, and more.

LSTM and GRU networks are designed to address the vanishing gradient problem in traditional recurrent neural networks, which enables them to capture long-term dependencies in sequential data, which make them a good tool for financial market metrics predictions and trend analysis.

Both LSTM and GRU networks incorporate memory cells to store and manage information across various time steps. This memory mechanism enables them to retain important patterns in the data and use that knowledge for future predictions. Additionally, the gating mechanisms in these networks enable them to control the flow of information, focusing on relevant features and filtering out noise or irrelevant data. This capability enhances their ability to learn significant patterns in the data and make precise predictions.

While LSTM and GRU networks have their limitations and require careful hyperparameter tuning, they are considered powerful tools in the field of deep learning, and their effectiveness has been demonstrated in numerous applications.

Multitasking in machine learning represents a compelling approach to addressing the challenges of complex and interconnected tasks. By leveraging shared information across tasks, multitasking can lead to improved efficiency, accuracy, and adaptability in various domains. However, successful multitasking requires careful consideration of task relationships, optimization techniques, and regularization strategies. As machine learning continues to advance, multitasking stands as a promising avenue for pushing the boundaries of AI capabilities and driving innovation across a wide range of applications [37].

The loss function acts as a compass, indicating the model performance on the task, while gradient descent acts as a navigator, steering the model towards parameter values that yield lower loss. Together, they form a dynamic duo that powers the model learning journey, making incremental adjustments to improve predictive accuracy and convergence.

In essence, the loss function shapes the landscape that gradient descent traverses, guiding the optimization process toward optimal model parameters that lead to enhanced performance. This symbiotic relationship underpins the essence of model training and optimization in machine learning.

Chapter 4

Deep Learning Applications to Stock Market Analysis

In the intricate realm of financial markets, the pursuit of accurate prediction and informed decision-making is an unending quest [10], [20], [22]. Over the years, deep learning has emerged as a formidable tool, revolutionizing the way we analyse and forecast market behaviours. This chapter serves as an introduction to the extensive literature review that embarks on a dual investigation: the application of deep learning for stock market prediction and the emerging paradigm of multitasking, which leverages the power of deep learning to address multiple interconnected tasks within this complex domain.

Deep learning, inspired by the architecture of the human brain, holds the promise of unravelling intricate patterns hidden within massive and intricate financial datasets. Within the context of stock market prediction, deep learning techniques hold the potential to decipher non-linear relationships, temporal dependencies, and latent variables that drive stock prices [41]. This introduction lays the foundation for a comprehensive journey into the mechanics of deep learning, encompassing neural network architectures, training strategies, and evaluation methodologies, all tailored to decode the intricate behaviours of financial markets.

The chapter covers the following key topics:

Stock Market Prediction: This literature review investigates the realm of deep learning in stock market prediction by examining existing research. It navigates through various studies, methodologies, and insights, dissecting the evolution of techniques from inception to cutting-edge models. The aim is to highlight the transformative potential of deep learning and multitasking in stock market analysis.

Multi-Task Deep Learning for Stock Market Analysis: Within this literature review, the research illuminates the multifaceted dimensions of stock market prediction and multitasking, showcasing how deep learning not only enhances predictive accuracy but also paves the way for tackling multiple interrelated tasks simultaneously. This includes addressing tasks such as stock price forecasting, volatility estimation, trend identification, and risk assessment, all while capitalizing on shared information and representations.

As the research journey through the subsequent chapters, it investigates thoroughly into the core studies, methodologies, and breakthroughs within the expansive landscape of deep learning for stock market prediction and multitasking. Through meticulous examination and critical analysis of existing literature, the aim is to extract insights that will illuminate new pathways toward more precise predictions, holistic perspectives, and empowered decision-making in the ever evolving and intricate arena of financial markets.

4.1 Deep Learning for Stock Market Prediction

The application of deep learning techniques in the field of market prediction has garnered substantial attention in recent years. This literature review aims to provide an overview of key studies and advancements in utilizing deep learning for forecasting market behaviours, with a focus on stock prices, market trends, and volatility.

Deep learning models have been extensively studied for stock price prediction. In a study [42], a hybrid model was introduced, integrating Long Short-Term Memory (LSTM) networks with a Reinforcement Learning (RL) framework for accurate stock price prediction. Additionally, deep neural networks were deployed in research [43] to capture intricate patterns in stock price data, demonstrating notable predictive performance.

Deep learning exhibits potential in recognizing market trends. In an extensive investigation, a comparison was made between the performance of deep learning models and traditional methods for predicting stock trends, illustrating the superiority of deep learning approaches [44]. This sentiment is reinforced by those who utilized convolutional neural networks (CNN) to extract meaningful features for precise trend identification [45].

Forecasting volatility is vital for effective risk management. A hybrid prediction model, which combines LSTM and GARCH, was introduced for cryptocurrency portfolio volatility in research [46], yielding superior results compared to conventional models. In an alternative strategy, a study [47] utilized LSTM with autoencoder models for forecasting foreign exchange volatility, highlighting deep learning's potential in capturing intricate volatility patterns.

Ensemble techniques that incorporate deep learning have gained prominence as well. A proposed ensemble model for stock market data prediction in study [48] showcases complementary strengths compared to other predictive models. Additionally, another research [49] introduced an ensemble of deep learning models for enhanced accuracy in stock market forecasting, significantly outperforming existing prediction models using the same dataset.

Transfer learning has emerged as a valuable approach. In a study [50], transfer learning was implemented on financial time series data, utilizing pre-trained neural networks to improve the accuracy of predicting stock price movements. This investigation underscores the versatility of deep learning techniques in adapting to the dynamics of financial markets.

The literature review underscores the growing significance of deep learning in market prediction. These studies collectively demonstrate the capacity of deep learning models to capture complex patterns, identify trends, and forecast volatility in financial markets. The integration of ensemble techniques, transfer learning, and innovative approaches such as multitasking and multimodal fusion highlights the dynamic nature of deep learning impact on market prediction.

A noteworthy contribution to this field is elucidated in **Chapter 5** of this thesis, which is partially published and references in [51]. A comprehensive methodology was adopted to study the Multivariate LSTM architecture, encompassing the integration of multiple input features, aimed at predicting the volatility of banking sector stocks. The model furnishes volatility projections for varying timeframes 1, 5, 10, and 20 days encompassing diverse market scenarios, encompassing both bullish trends characterized by ascending prices and bearish trends marked by descending prices. The outcomes of the research uncover a notable trend wherein the predictions generated by the Multivariate LSTM model outperform those of the GARCH model, corroborating earlier investigations. The incorporation of an array of input features amplifies the model predictive prowess, thereby underscoring the efficacy of this approach in bolstering volatility forecasts within the financial landscape.

4.2 Multi-Task Deep Learning for Stock Market

The integration of multi-task deep learning into stock market analysis has gained significant traction, offering a powerful framework to simultaneously address diverse aspects of market behaviour and enhance predictive capabilities. This literature review provides an overview of key studies and advancements in utilizing multi-task deep learning for stock market prediction, volatility forecasting, and risk management.

The utilization of multi-task deep learning architecture extends to diverse stock market prediction tasks. In a study by [52], a multi-task deep learning model was introduced, simultaneously addressing stock price prediction and financial risk assessment. By capitalizing on shared information, this innovative approach enhances the accuracy of predictions for both tasks. The research underscores the advantages of capturing interdependencies between tasks, thereby augmenting the overall predictive performance.

Additionally, multi-task deep learning has found application in the realms of risk evaluation and portfolio management. A noteworthy example is presented in research [53], where they a multi-task framework was devised within a deep neural network structure. This innovative approach involves the simultaneous acquisition of insights into portfolio construction alongside various auxiliary tasks linked to volatility. Notably, the model excels in forecasting realized volatility, gauged through diverse volatility estimators, thereby surpassing established strategies. In reference [54] an innovative multi-task strategy was introduced, leveraging domain-specific textual attributes and precise audio alignment to forecast financial risk and price behaviour. their approach enhances current methodologies along two dimensions: firstly, by customizing a sophisticated deep multimodal model that focuses on both text and audio inputs, and secondly, by enhancing the prediction of volatility and price movements through a comprehensive ensemble formulation that addresses multiple tasks simultaneously.

Transfer learning and multimodal fusion techniques have been integrated into multi-task deep learning for enhanced stock market analysis. In study [55], transfer learning was leveraged to enable knowledge transfer between related financial tasks, leading to improved predictive performance in comparison to single task model. Moreover, multimodal deep learning models have been extended to incorporate scatter plots to overcome the complexity of cross-correlation between domestic and foreign markets to further enriching the predictive capabilities of the model [56].

Multi-Task deep learning has demonstrated efficacy in capturing temporal dependencies in time series data. In the research [57], a multi-task architecture was introduced for stock price prediction and temporal trend analysis, showcasing the potential of joint learning to capture evolving market dynamics during extreme market condition such as one experienced during COVID-19.

While multi-task deep learning offers promising avenues for stock market analysis, challenges include task weighting, optimal model architecture design, and potential overfitting. Adequate task weighting and selection are crucial to ensure that each task contributes meaningfully to the overall learning process.

The reviewed literature underscores the growing significance of multi-task deep learning in stock market analysis. These studies collectively emphasize the potential of joint learning to capture interdependencies among various financial metrics, leading to improved prediction accuracy and enhanced risk management strategies. The integration of transfer learning, multimodal fusion, and temporal analysis further underscores the versatility and dynamism of multi-task deep learning approaches in the domain of stock market analysis.

In **Chapter 6** of this thesis, an inventive solution is detailed for predicting multiple crucial stock market metrics through the application of multi-task deep learning. The study is conducted on the S&P 500 and Dow Jones indexes across varying market conditions—bull markets, bear markets, and volatile periods deploying prominent deep learning networks within this domain, namely LSTM and GRU. To ensure comprehensiveness, the predictive accuracy is benchmarked against well-established statistical models, GARCH and ARIMA, and juxtaposed with a single-task model for future predictions spanning 1, 5, 10, and 20 days.

The observed outcomes validate that the multi-task model capitalizes on the shared layer and dataset deployed across all tasks, thereby positioning it as superior to its single-task counterpart and highly competitive when compared to existing statistical models.

4.3 Conclusions

In the realm of financial markets, the integration of deep learning and multitasking stands as a transformative force. Deep learning, drawing inspiration from neural networks and the human brain, has emerged as a potent tool for decoding complex market behaviours. Its application spans across stock price prediction, volatility estimation, and trend identification, offering nuanced insights into intricate relationships within financial data.

Multitasking, on the other hand, introduces a paradigm shift by enabling a single model to concurrently tackle multiple related tasks. This approach leverages shared knowledge and representations, enhancing the model ability to decipher interdependencies and uncover hidden patterns within the market. By jointly optimizing parameters across tasks, multitasking leads to improved generalization, enhanced accuracy, and robustness in predictions.

In the financial domain, this synergy of deep learning and multitasking offers a holistic approach. It empowers traders and investors to address a multitude of market analysis dimensions, from stock price movements to risk assessment, all within a unified framework. The integration of advanced technologies and algorithms further amplifies this approach, automating tasks and delivering real-time insights for informed decision-making.

In essence, the fusion of deep learning and multitasking in financial markets redefines predictive capabilities, paving the way for more precise forecasts, improved trading strategies, and ultimately, enhanced performance within the dynamic and intricate world of finance.

Chapter 5

Deep Learning for Stock Market Prediction: Unveiling Insights and Enhancing Forecasts

Participating in the stock market necessitates an understanding of financial exchanges and the assets being traded. Additionally, it entails embracing specific investment strategies and approaches chosen thoughtfully to maximize gains and minimize potential losses, it also demands ongoing recognition and control of various investment prospects and the corresponding risks they entail [10], [13].

Effective risk management plays a crucial role in achieving success with investment and trading strategies. Diverse categories of risks exist, capable of negatively influencing an investment or trading collection, necessitating the application of varied approaches to reduce and safeguard against them. Figure 1 provides a visual representation of common probabilities that can result in losses in returns [13].

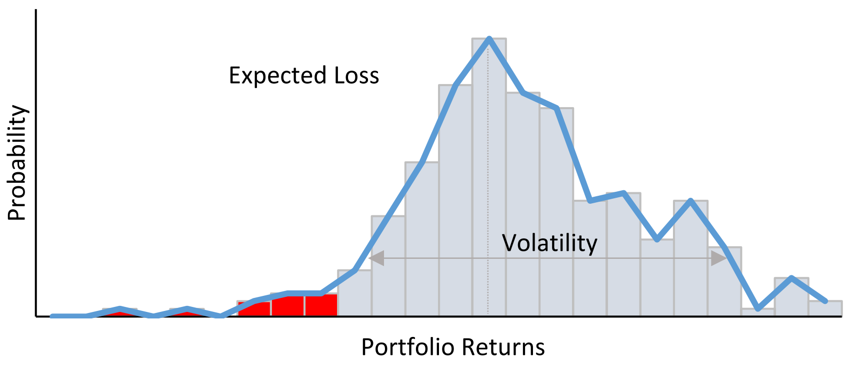


Figure 5.1: Portfolio loss probability. JP Morgan daily returns, 2017.

Volatility serves as a quantification of the oscillation observed in the price of a security or a market index, playing a fundamental role in assessing risk and functioning as a pragmatic tool for navigating uncertainty within financial markets [9]. A variety of volatility categories exist, encompassing a security volatility in relation to a benchmark index (Beta), volatility derived from historical price movements (Historical Volatility), and the projected volatility spanning an option lifespan (Implied Volatility), as detailed in references [18] and [15]. The latter, Implied Volatility, finds application in the renowned Black-Scholes formula for pricing options [19].

The realm of volatility forecasting has engaged the attention of both industry practitioners and scholarly researchers for numerous decades. Enhanced precision in predicting market volatility yields superior risk management capabilities and refined pricing models, thereby facilitating the implementation of strategies geared toward maximizing profitability in trading and investments. In contemporary financial practices, statistical models such as GARCH [3] and [24] typically assume the role of forecasting volatility and deciphering price movements in stock markets. Given the substantial surge in trading volumes and the amplification of market-impacting factors in recent times, attributed to the rapid advancements in internet technologies and mobile networks, an appetite for alternative approaches to model volatility has emerged. This demand seeks to augment processing speed and accuracy when handling extensive datasets characterized by intricate structures.

Notably, the spotlight has turned towards Artificial Neural Networks (ANN) and Deep Learning methodologies, as these have garnered considerable attention due to their algorithmic advancements. These techniques hold potential to address the aforementioned challenges and further enrich the arsenal of tools available for volatility modelling and analysis.

The prevalence of extensively available vast datasets, coupled with the advancements in computing hardware such as specialized TensorFlow cores designed to handle substantial data volumes, has paved the way for utilizing specific algorithmic approaches.

This study investigates the application of a distinct variant of Recurrent Neural Networks (RNN) for the purpose of volatility prediction. RNNs, a type of feed-forward artificial neural network, deploy loops that enable the retention of information, thus furnishing a memory mechanism capable of capturing sequential patterns in time series data [25]. Notably, this investigation adopts the utilization of LSTM networks [27], an exceptionally potent subset of RNNs.

Within this chapter, the primary aim of this study is elucidated – to gauge the efficacy of a Multivariate LSTM architecture. This assessment involves a comparative analysis of its predictive precision in relation to two benchmarks: the GARCH model, and a singular-input deep learning architecture.

The aim of this research is to validate the first two hypotheses outlined in this thesis: (1) The deployment/use of deep learning for market volatility prediction yields outcomes matches or outperforms those of statistical models such as GARCH; (2) The utilization of stacked multivariate deep learning with a composite input consisting of multiple stock prices enhances the predictive accuracy of a key stock market performance indicator (future realized volatility is selected for this research).

To evaluate the model efficacy, a series of tests were conducted utilizing a custom-built tool in Python, deploying TensorFlow libraries.

The subsequent sections of this chapter are structured as follows:

Section 5.1: Section 5.1 offers a comprehensive literature review, examining prior research pertaining to the utilization of deep learning for volatility prediction within the financial domain.

Section 5.2: Section 5.2 elucidates the methodology adopted for constructing the multivariate deep learning model, expounding on the testing procedures, variables deployed, dataset specifics, and the array of conducted experiments.

Section 5.3: Section 5.3 delineates the outcomes derived from benchmark testing, coupled with a comparative analysis of single-input and multivariate deep learning models.

5.1 Background and Related Works

Various methodologies are currently deployed to model market dynamics and volatility. Recent years have witnessed an intensified drive for reassessment and enhancement of prevailing pricing models and risk management strategies, propelled by the influx of extensive data volumes and the lingering apprehensions stemming from the 2008 financial crisis [1], [2].

Numerous studies have juxtaposed Artificial Neural Networks (ANN) with statistical models such as GARCH for the purpose of forecasting volatility, consistently revealing that ANN can offer heightened effectiveness and accuracy. The rationale behind this preference rests in the intricate characteristics of ANN – their propensity to handle high non-linearity, continuous data, large-scale datasets, temporal dependencies, and dynamic patterns [58]– making them a compelling choice for time series prediction within this specific domain.

In the study [59], an incisive examination was conducted into the computational prowess of ANN for modelling time-varying data. Their study underscored that ANN not only align with conventional time-series theory but also hold potential as robust alternatives to established models, particularly in cases where non-linearity plays a pivotal role. Similarly, the research in [60] corroborated the time-series capabilities of ANN while underscoring the significance of the sampling window size. Their findings illuminated that optimal performance materializes when the appropriate embedding dimension is maintained, with deviations from this window size leading to performance degradation.

In a parallel undertaking, [61] executed an empirical study aimed at forecasting the Japanese Nikkei 255 index. The study evaluated the classical Back Propagation Neural Network (BPNN) against Genetic Algorithm (GA) and Simulated Annealing (SA), ultimately concluding that the fusion of classical BPNN with global search techniques enhances both accuracy and speed. Notably, other investigations have also showcased promising outcomes in stock price prediction through the utilization of Deep Learning methodologies [7].

In the study outlined in [62], researchers deployed a Wavelet De-noising-based Back Propagation (WDBP) neural network. Within this model, the initial dataset underwent wavelet transform-based decomposition, resulting in multiple layers of data representation.

In a separate investigation detailed in [63], the implementation of Long Short-Term Memory (LSTM) was deployed using a dataset spanning 23 years of SP500 daily index prices, with sequences of 240 days. The findings indicated that LSTM exhibited superior predictive capabilities in comparison to random forest. The study concluded that LSTM can effectively facilitate the formulation of profitable trading strategies. In [64], the researchers combined a multi-layer LSTM architecture with Empirical Mode Decomposition (EMD) to enhance time series prediction. The resultant model showcased enhanced forecasting aptitude when contrasted with the Support Vector Machine (SVM). Similarly, the study presented in [65] introduced an LSTM-based model engineered to prevent overfitting while predicting the SP500. The outcomes showcased a commendable level of forecasting accuracy.

In [66], Deep LSTM (DLSTM), a specialized architecture derived from LSTM, was proposed as a potent tool for time series modelling. Comparative assessments against the Autoregressive

Integrated Moving Average (ARIMA) model confirmed the superiority of DLSTM. While the research examined various lagging configurations, no definitive evidence for an optimal setup emerged.

A stacked LSTM architecture to model time series data has been used in many studies, the research in [67] exhibited significant advancements in prediction performance compared to single LSTM models, while the research conducted in [68] echoed these findings and emphasized the heightened accuracy of this approach over traditional statistical methodologies in time series modelling.

In an innovative approach presented by [69], the fusion of traditional Artificial Neural Networks (ANN) with LSTM and bidirectional LSTM (BLSTM) was conducted to refine time series forecasting accuracy. Results indicated the superior performance of BLSTM for predicting volatility three weeks ahead, while LSTM demonstrated proficiency in shorter time horizons. [70] adopted a hybrid model combining LSTM and traditional ANN techniques to enhance gold volatility forecasting. Through experiments deploying varied lagging setups to facilitate temporal learning, this hybrid model showcased heightened accuracy compared to classic GARCH and standalone LSTM models.

5.2 Methodology

Multiple experiments were carried out to evaluate the efficiency of a multivariate deep learning architecture. The aim was to compare its predictive accuracy against single-input models and the traditional statistical model, GARCH. These experiments encompassed the application of diverse deep learning architectures and statistical models to gauge their precision in forecasting various stock market volatilities. These evaluations are crucial not only for substantiating and examining "Hypothesis 1" and "Hypothesis 2," which pertain to the efficacy of deep learning in stock market volatility prediction and its potential to enhance or replace existing models, but also for assessing the effectiveness of the multivariate deep learning architecture in enhancing the performance of deep learning.

5.2.1 Testing Modules and Metrics

The research involved multiple experiments focused on different stock market volatility for key global banks stocks, utilizing statistical and deep learning models. The metrics used to test performance is the Mean Squared Error (MSE) between actual volatility and predicted volatility in different market condition and different future prediction horizons.

Three main test modules were created to achieve this:

GARCH Module: This module expects a single input variable representing the stock market matric. The input is fed into a GARCH model, subsequently contrasting the resultant output (realized volatility) with the factual volatility. The performance of the model is gauged using MSE.

Single-Input Deep Learning Module: This implementation expects single input variables representing a stock market metric. It feeds the input to the model, and a single output is compared against actual data. MSE is used to measure the model accuracy.

Multivariate Deep Learning Module: This implementation anticipates numerous input variables that signify an individual stock or an assemblage of stocks from the stock market. These inputs are then supplied to the model, with a singular output being juxtaposed against real data. The accuracy of the model is assessed using MSE.

5.2.2 Data Collection

The yfinance Python 3.9 Application Programming Interface (API) from Yahoo Finance (https://finance.yahoo.com) was deployed to fetch the historical daily closing prices for a specific set of stocks. Two different time periods were selected: a period when the market was experiencing an upward trend (bull market) and a period when it was in a downward trend (bear market). Furthermore, data for oil prices was obtained from the U.S. Energy Information Administration (www.eia.gov), whereas gold prices were acquired from the World Gold Council (https://www.gold.org).

Two distinct window horizons were gathered from varying market conditions, serving as stress tests for the models. This approach aimed to evaluate their performance under both short and extended training windows.

Comprehensive information regarding the data deployed in the examinations is presented in Table 5.1 and Table 5.2.

Table 5.1: Test Data Definition – Stock Market Symbols.

Symbol	Company	Sector
BAC	Bank of America	Financial Services
JPM	JP Morgan Chase	Financial Services
WFC	Wells Fargo	Financial Services
C	City Bank	Financial Services
DB	Deutsche Bank	Financial Services
MS	Morgan Stanley	Financial Services
GS	Goldman Saches	Financial Services
WMT	Walmart Inc	Consumer Defensive
IBM	IBM	Technology

Table 5.2: Test Data Definition – Timeseries.

Market Condition	Name	Sample Size	Window
Bear	Long	2304	03/01/2000 - 03/03/2009
Bull	Long	4467	03/01/2000 - 01/10/2017
Bear	Short	603	03/08/2006 - 29/01/2009
Bull	Short	607	26/08/2014 - 31/01/2017

5.2.3 Experimental Procedures

During the testing phase, Multivariate and Single-Input learning Recurrent Neural Network (RNN) architectures were constructed using Python 3.7 and TensorFlow 2.0. In order to carry out comparative evaluations, a GARCH statistical model was coded utilizing the arch and other Python 3.7 libraries. All experimentation was conducted within a consistent testing environment and application context.

Application structure used in testing is depicted in Figure 5.2. Sample pseudocode demonstrating the calculation of daily returns and volatility is presented in Appendix D for reference.

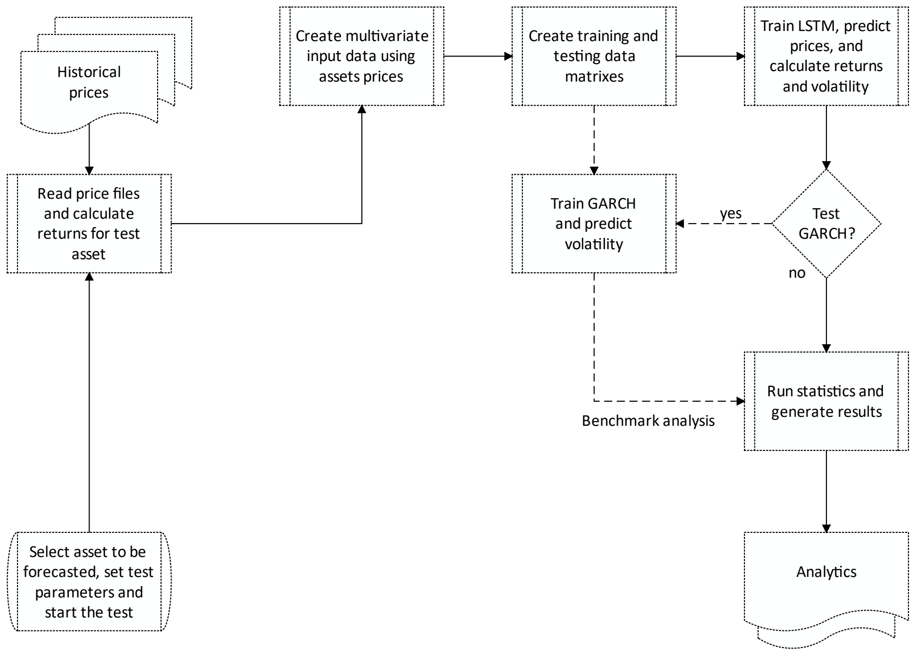

Historical prices
Create multivariate input data using assets prices
Create training and testing data matrixes
Train LSTM, predict prices, and calculate returns and volatility
Read price files and calculate returns for test asset
Train GARCH and predict volatility
yes
Test GARCH?
no
Benchmark analysis
Run statistics and generate results
Select asset to be forecasted, set test parameters and start the test
Analytics

Figure 5.2: Testing Application.

In each trial, the hold-out performance estimation technique was deployed. The training data for both the Deep Learning networks and the statistical model consisted of historical daily stock prices, daily gold prices, and daily crude oil prices. The experiments encompassed a range of market scenarios, including bear and bull markets, to comprehensively evaluate the model performance across diverse conditions. To ensure robustness, each test was iterated 10 times, resulting in comprehensive outcomes. These test outcomes include both the average Mean Squared Error (MSE) and the standard deviation of MSE, providing a comprehensive analytical perspective.

The details of the server, GPU card, and other computational resources utilized for the entire testing process are listed in Table 5.3 and Table 5.4.

Table 5.3: CPU specification.

Server Specifications	
CPU	Intel Xeon CPU E5-2640 @3.0GHz (2 processors, 24 cores)
Memory	32 GB
Environment	UBUNTU 18.04
Language	Python 3.7
ML Framework	TensorFlow 2.0, GPU

Table 5.4: GPU specifications.

GPU Specifications	
GPU	GeForce RTX 2060 SUPER
Memory	8 GB
CUDA Cores	2176
GPU Clock	1650 MHz

5.2.3.1 Evaluation through Benchmark Testing

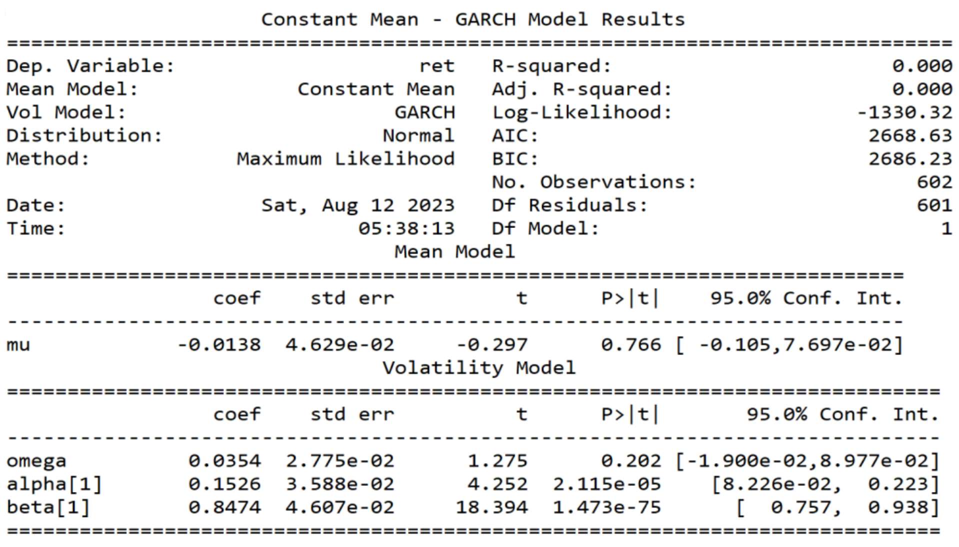
The purpose of these experiments is to contrast the accuracy of Deep Learning models with the reference benchmark statistical model, GARCH. This comparison is essential for providing support to the core of "Hypothesis 1".

The executed tests for both GARCH and DL models are elaborated upon in Table 5.5. Each test is executed using data obtained from both bull and bear market scenarios.

Table 5.5: Benchmark - Input and output for GARCH and DL models tests.

Model	Input/ Output	Prediction Days	Trials	Market Condition	Window
GARCH, LSTM	BAC	1, 5, 10, 20	10	Bear, Bull	Short, extended

In the process of training the GARCH model, the daily returns were computed using Equation 2.10, and the data was normalized using the training dataset. Following that, the testing dataset was scaled using the training data before being deployed for model evaluation. A representative summary of the GARCH model can be seen in Figure 5.3, which was utilized to forecast the 5-day rolling realized volatility of BAC. The procedure for training and fitting the GARCH model is depicted in Code 5.1. Once the model is trained, it is applied to the test data for volatility prediction.



```
                     Constant Mean - GARCH Model Results
==============================================================================
Dep. Variable:                    ret   R-squared:                       0.000
Mean Model:             Constant Mean   Adj. R-squared:                  0.000
Vol Model:                      GARCH   Log-Likelihood:               -1330.32
Distribution:                  Normal   AIC:                           2668.63
Method:            Maximum Likelihood   BIC:                           2686.23
                                        No. Observations:                  602
Date:                Sat, Aug 12 2023   Df Residuals:                      601
Time:                        05:38:13   Df Model:                            1
                                 Mean Model
==========================================================================
                 coef    std err          t      P>|t|      95.0% Conf. Int.
--------------------------------------------------------------------------
mu            -0.0138  4.629e-02     -0.297      0.766 [ -0.105,7.697e-02]
                              Volatility Model
=============================================================================
                 coef    std err          t      P>|t|         95.0% Conf. Int.
-----------------------------------------------------------------------------
omega          0.0354  2.775e-02      1.275      0.202 [-1.900e-02,8.977e-02]
alpha[1]       0.1526  3.588e-02      4.252  2.115e-05    [8.226e-02,  0.223]
beta[1]        0.8474  4.607e-02     18.394  1.473e-75      [  0.757,  0.938]
=============================================================================
```

Figure 5.3: Benchmark - GARCH model summary.

Code 5.1: Benchmark - GARCH model fitting logic.

```
rolling_predictions ← List()
**for** i from 1 to test_data.size() **do**
    train_data ← returns[:i - test_data.size()]
    model = garch(tarain_data)
    model_fit ← model.fit(p, o, q)
    prediction ← forecast GARCH model_fit(horizon ← days_to_predict)
    predictions.insert(sqrt(prediction.variance))
**end for**
```

A comparable strategy was deployed during the testing phase involving the LSTM DL model. The data underwent scaling using the training dataset, and subsequently, the predicted testing data was scaled back accordingly. The hyperparameters utilized for the LSTM model are presented in Table 5.6 which resulted from a similar code depicted in Code 5.1. The visual representation of the LSTM model diagram for forecasting BAC 5-day rolling realized volatility can be observed in Figure 5.4.

Table 5.6: Benchmark - LSTM hyperparameters.

Epochs	Neurons	Batch Size	Optimization Algorithm
100	64	100	RMSProp

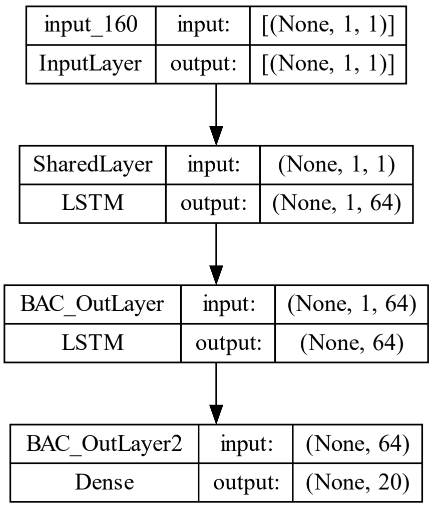


Figure 5.4: Benchmark – LSTM model diagram.

5.2.3.2 Comparing Multivariate and Single-Input Models

The set of experiments has been meticulously planned to validate the central hypothesis of this research. The primary objective is two-fold: first, increase the accuracy of Multivariate Deep Learning (MDL) models when juxtaposed with Single-Input Deep Learning (SDL) models, and secondly, to accentuate the merits inherent in adopting the MDL architecture. It worth noting that this MDL architecture forms the crux of “Hypothesis 2” that this study revolves around.

These experiments critically substantiate the core findings of this research study. By empirically demonstrating the superior accuracy of MDL models over their SDL counterparts, the experiments contribute to the broader understanding of the effectiveness of advanced neural network architectures in time series analysis. Furthermore, the emphasis on the advantages brought by the MDL architecture underscores the significance of examining innovative approaches in the pursuit of more accurate and reliable predictive modelling techniques.

Through meticulous experimentation and comparison, this series of tests serves to provide substantial evidence to support the central claims made by this research. The conclusive findings will contribute valuable insights to the field and offer a robust foundation for further advancements in time series prediction methodologies.

To ensure an equitable comparison between the models, the hyperparameters deployed for both the Multivariate Deep Learning (MDL) and Single-Input Deep Learning (SDL) models were deliberately kept identical. This strategic decision was made to eliminate any potential bias that could favour one model over the other due to disparate hyperparameter settings. By maintaining consistent hyperparameters, the comparison becomes more valid and enables for a more accurate assessment of the models' intrinsic capabilities.

It is important to highlight that this approach aligns with recommended practices in experimental design, as it minimizes the influence of external factors and enables a focused evaluation of the models' performance differences. The choice to match hyperparameters underscores the research commitment to a rigorous and unbiased analysis of the MDL and SDL models.

For a comprehensive understanding of the specific hyperparameters deployed in both the MDL and SDL models, the complete details are meticulously documented in Table 5.7.

Table 5.7: MDL and SDL hyperparameters.

Epochs	Neurons	Batch Size	Output (Days)	Optimization Algorithm
100	8	1000	1, 5, 10, 20	RMSProp

The LSTM model utilized for Single-Input Deep Learning (SDL) is identical to the one deployed in the benchmark testing, as depicted in Figure 5.4 with neurons equal to 8 instead of 64. In contrast, distinct models were constructed for the MDL experiments. These MDL models were designed to accommodate diverse combinations of multivariate inputs, serving the purpose of rigorously validating the hypothesis. This validation was achieved by systematically evaluating the impact of deploying varied input combinations, encompassing different numbers of correlated and non-correlated instruments drawn from disparate sectors. A comparative analysis with the SDL model was then carried out.

An exemplar of the MDL architecture designed to predict BAC realized volatility utilizing three inputs is visually represented in Figure 5.5. This specific architecture configuration serves to provide a tangible illustration of the underlying method.

For a comprehensive understanding of the full spectrum of MDL tests undertaken, a comprehensive list is documented in Table 5.8. This list encapsulates the variations in input combinations, each aimed at uncovering insights into the efficacy of the MDL approach in comparison to the SDL model across diverse scenarios and conditions.

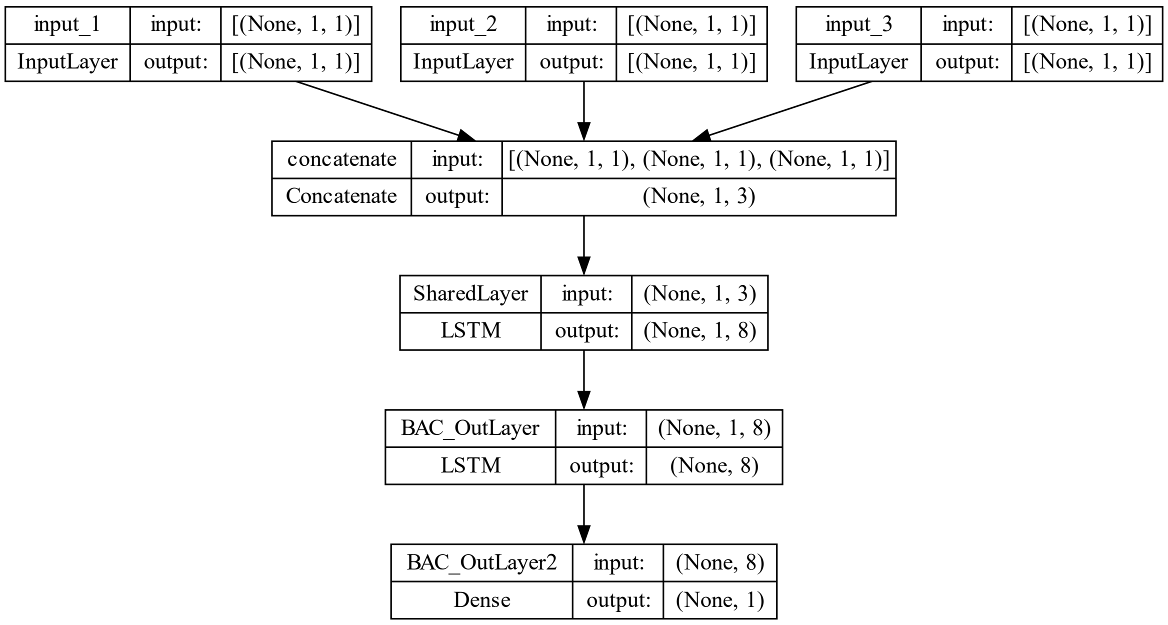

input_1 | input: | [(None, 1, 1)]
InputLayer | output: | [(None, 1, 1)]
input_2 | input: | [(None, 1, 1)]
InputLayer | output: | [(None, 1, 1)]
input_3 | input: | [(None, 1, 1)]
InputLayer | output: | [(None, 1, 1)]
concatenate | input: | [(None, 1, 1), (None, 1, 1), (None, 1, 1)]
Concatenate | output: | (None, 1, 3)
SharedLayer | input: | (None, 1, 3)
LSTM | output: | (None, 1, 8)
BAC_OutLayer | input: | (None, 1, 8)
LSTM | output: | (None, 8)
BAC_OutLayer2 | input: | (None, 8)
Dense | output: | (None, 1)

Figure 5.5: Benchmark – LSTM model diagram.

The model optimization process involves assessing its trainable parameters and adjusting them to align with various testing requirements. Equation 7.1 is used to calculate and every new model before adjusting hyperparameters [11], [12].

If there is an abundance of parameters, there is an increased risk of overfitting, while a scarcity of parameters may lead to suboptimal performance.

$$LSTM\ Prams_{No.} = 4\left(\left(i+h\right)x\ h+h\right)$$

$$GRU\ Params_{No.} = 3\left(\left(i+h\right)x\ h+\left(2\ x\ h\right)\right)$$

$$RNN\ Params_{No.} = \left(\left(i+h\right)x\ h\right)+\ h$$

Equation 5.1: Number of parameters used in DL network formulas.

Where:

i the number of inputs.

h is the number of hidden networks.

Table 5.8: Test 2 – LSTM model parameters.

Model	Input	Output	Prediction Days	Trials	Market Condition	Window
SDL	BAC	BAC	1, 5, 10, 20	10	Bear, Bull	Long, short
MDL	BAC, C	BAC	1, 5, 10, 20	10	Bear, Bull	Long, short
MDL	BAC, DB	BAC	1, 5, 10, 20	10	Bear, Bull	Long, short
MDL	BAC, GOLD	BAC	1, 5, 10, 20	10	Bear, Bull	Long, short
MDL	BAC, GS	BAC	1, 5, 10, 20	10	Bear, Bull	Long, short
MDL	BAC, IBM	BAC	1, 5, 10, 20	10	Bear, Bull	Long, short
MDL	BAC, JPM	BAC	1, 5, 10, 20	10	Bear, Bull	Long, short
MDL	BAC, MS	BAC	1, 5, 10, 20	10	Bear, Bull	Long, short
MDL	BAC, OIL	BAC	1, 5, 10, 20	10	Bear, Bull	Long, short
MDL	BAC, WMT	BAC	1, 5, 10, 20	10	Bear, Bull	Long, short
MDL	BAC, DB, C	BAC	1, 5, 10, 20	10	Bear, Bull	Long, short
MDL	BAC, DB, MS	BAC	1, 5, 10, 20	10	Bear, Bull	Long, short
MDL	BAC, GOLD, DB	BAC	1, 5, 10, 20	10	Bear, Bull	Long, short
MDL	BAC, GOLD, IBM	BAC	1, 5, 10, 20	10	Bear, Bull	Long, short
MDL	BAC, GOLD, WMT	BAC	1, 5, 10, 20	10	Bear, Bull	Long, short
MDL	BAC, GS, C	BAC	1, 5, 10, 20	10	Bear, Bull	Long, short
MDL	BAC, GS, MS	BAC	1, 5, 10, 20	10	Bear, Bull	Long, short
MDL	BAC, JPM, C	BAC	1, 5, 10, 20	10	Bear, Bull	Long, short
MDL	BAC, JPM, DB	BAC	1, 5, 10, 20	10	Bear, Bull	Long, short
MDL	BAC, JPM, GS	BAC	1, 5, 10, 20	10	Bear, Bull	Long, short
MDL	BAC, JPM, MS	BAC	1, 5, 10, 20	10	Bear, Bull	Long, short
MDL	BAC, MS, C	BAC	1, 5, 10, 20	10	Bear, Bull	Long, short
MDL	BAC, OIL, DB	BAC	1, 5, 10, 20	10	Bear, Bull	Long, short
MDL	BAC, OIL, IBM	BAC	1, 5, 10, 20	10	Bear, Bull	Long, short
MDL	BAC, OIL, JPM	BAC	1, 5, 10, 20	10	Bear, Bull	Long, short
MDL	BAC, OIL, WMT	BAC	1, 5, 10, 20	10	Bear, Bull	Long, short
MDL	BAC, DB, GS, C	BAC	1, 5, 10, 20	10	Bear, Bull	Long, short
MDL	BAC, DB, GS, MS	BAC	1, 5, 10, 20	10	Bear, Bull	Long, short
MDL	BAC, GS, MS, C	BAC	1, 5, 10, 20	10	Bear, Bull	Long, short
MDL	BAC, JPM, GS, C	BAC	1, 5, 10, 20	10	Bear, Bull	Long, short
MDL	BAC, JPM, GS, DB	BAC	1, 5, 10, 20	10	Bear, Bull	Long, short
MDL	BAC, JPM, GS, MS	BAC	1, 5, 10, 20	10	Bear, Bull	Long, short
MDL	BAC, JPM, GS, DB, C	BAC	1, 5, 10, 20	10	Bear, Bull	Long, short
MDL	BAC, JPM, GS, DB, MS	BAC	1, 5, 10, 20	10	Bear, Bull	Long, short
MDL	BAC, JPM, GS, DB, C, MS	BAC	1, 5, 10, 20	10	Bear, Bull	Long, short

5.3 Results

The presentation of the results adheres to the systematic experimental protocols delineated within their respective sections. The sequence of conducted tests holds particular significance, as outcomes from specific tests play a role in shaping the results of other test sets. This deliberate arrangement holds strategic importance, as it facilitates the extraction of meaningful conclusions and a holistic evaluation of the extent to which the core objectives have been successfully attained. By establishing a structured flow between interrelated experiments, this approach ensures that the resulting analysis is both comprehensive and coherent, thereby contributing to a well-rounded understanding of the study achievements.

The full list of charts produced by this test is included in Appendix A, Section 2.1.

5.3.1 Evaluation through Benchmark Testing

Both a GARCH (1,1) model and a stacked Long Short-Term Memory (LSTM) model, utilizing a single input without auxiliary assets, were harnessed to forecast forthcoming realized volatility for Bank of America stock (BAC). To rigorously evaluate the models and ensure a robust analytical comparison, these tests were iterated across both short and long windows, systematically subjecting the models to varied temporal horizons.

When examining the outcomes of the extended window testing, which commences from the year 2000 as demonstrated in Table 5.9, the findings decisively establish the superiority of LSTM DL model over the GARCH model across all future prediction durations and market scenarios.

Table 5.9: Model accuracy - GARCH vs MDL – Long time window ($\mathbf{10^{-5}}$).

MSE				
	Bear Market		Bull Market	
Prediction (Days)	GARCH	LSTM	GARCH	LSTM
1	59.30	0.58	3.57	0.00
5	59.28	3.40	3.67	0.06
10	59.25	8.64	3.81	0.12
20	59.20	11.34	4.12	0.20

Furthermore, a notable trend emerges from the results: the LSTM model adeptly captures the evolving patterns within the data for each of the prediction days, particularly under extreme market conditions. In contrast, the GARCH model appears to exhibit a tendency to converge around the average point of the testing data, exhibiting a relatively uniform behaviour regardless of the specific future prediction horizon. This divergence in behaviour between the two models underscores the dynamic capabilities of the LSTM model in capturing nuanced changes within the data, particularly when facing challenging market conditions.

While inspecting the outcomes of the brief window analysis as presented in Table 5.10, a recurrence of the previously described behaviour is evident, particularly in bear market conditions. This observation reinforces the initial inference drawn from the results.

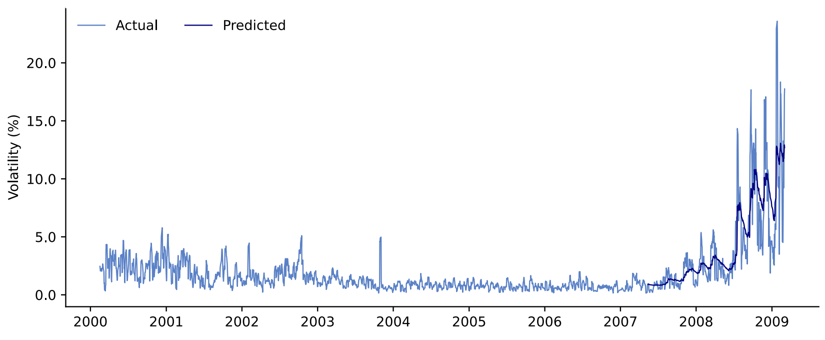
Table 5.10: Model accuracy - GARCH vs MDL – Short time window ($\mathbf{10^{-5}}$).

MSE				
	Bear Market		Bull Market	
Prediction (Days)	GARCH	LSTM	GARCH	LSTM
1	74.79	70.01	2.44	1.20
5	74.86	90.71	3.05	1.04
10	74.96	89.62	3.80	1.96
20	75.16	143.00	5.00	2.39

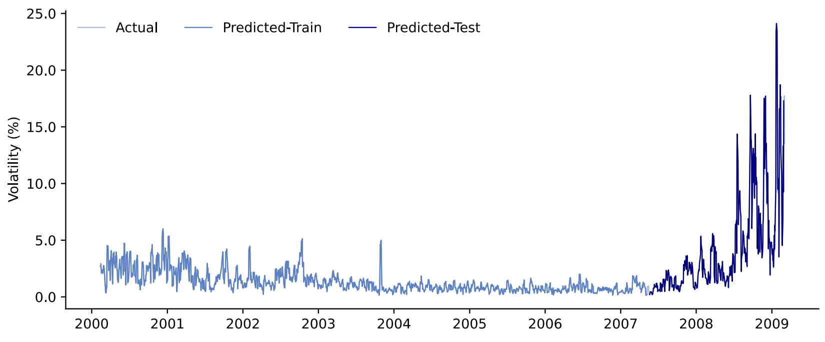
To visually interpret the aforementioned outcomes, charts depicting the performance of the GARCH and LSTM models for 1-Day future prediction of BAC volatility in both bear and bull markets, across both long and short windows, are presented in the figures below. A discernible pattern emerges from Figure 5.6 and Figure 5.8, where it becomes evident that GARCH exhibits a comparatively weaker performance in bear markets as opposed to LSTM. Notably, GARCH demonstrates relatively improved performance in short windows in comparison to long ones.

Shifting focus to bull markets, as depicted in Figure 5.7 and Figure 5.9, GARCH showcases a performance level that closely approximates that of LSTM. Despite this, the overarching performance superiority remains with LSTM. Interestingly, GARCH exhibits an aptitude for capturing volatility spikes more effectively, which is particularly evident in the visualization.

These visual representations provide an accessible means of grasping the observed trends, substantiating the analytical findings derived from the comprehensive experiments.

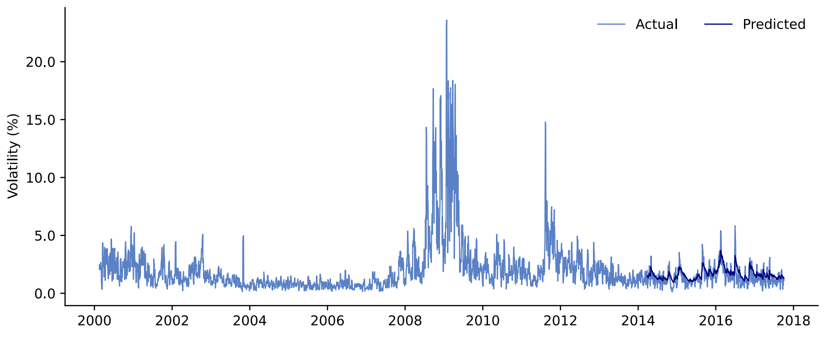


GARCH Model.

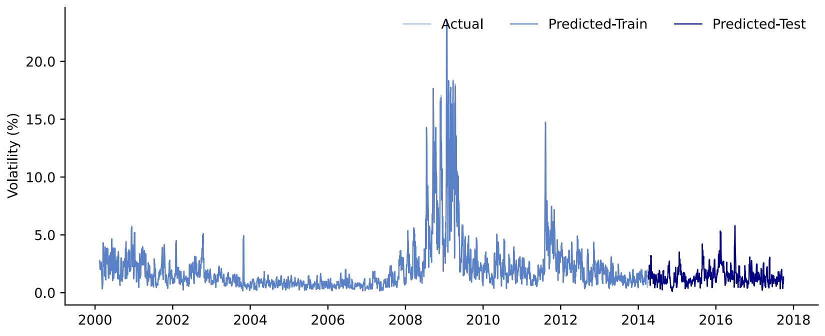


Deep Learning Model.

Figure 5.6: GARCH vs LSTM - Volatility prediction for BAC in bear market.

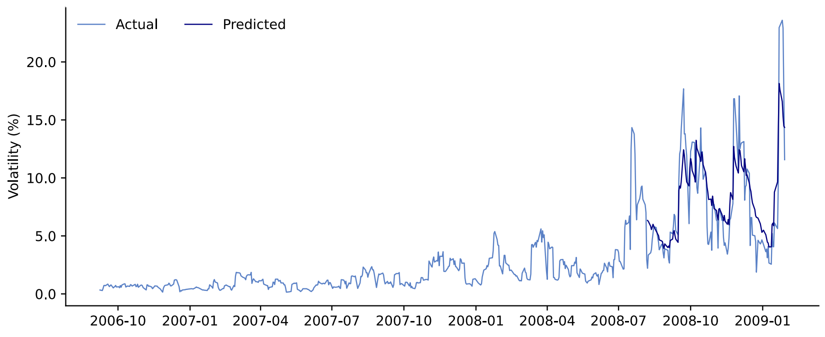


GARCH Model.

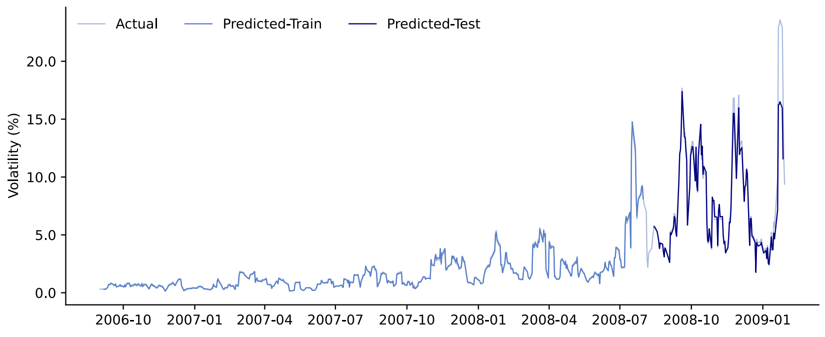


Deep Learning Model.

Figure 5.7: GARCH vs MDL - Volatility prediction for BAC in bull market.

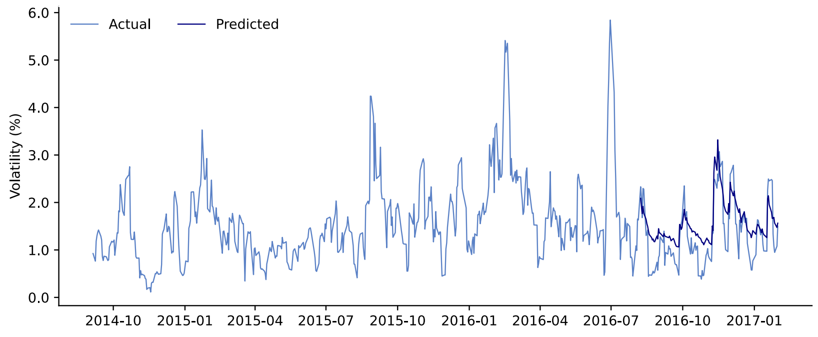


GARCH Model.

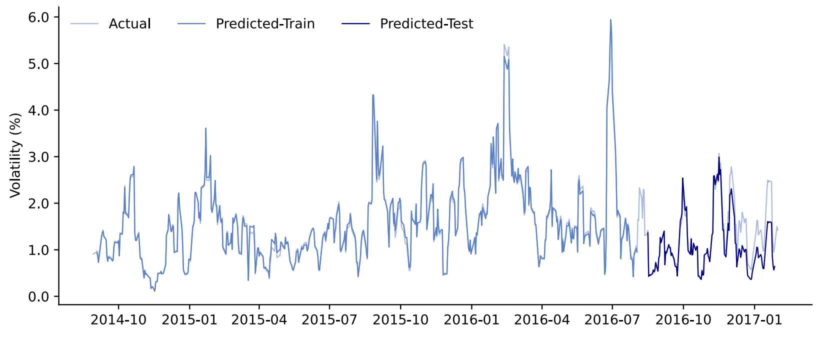


Deep Learning Model.

Figure 5.8: GARCH vs MDL - Volatility prediction for BAC in bear market.



GARCH Model.



Deep Learning Model.

Figure 5.9: GARCH vs MDL - Volatility prediction for BAC in bull market.

Moving forward, the focus shifts to the assessment of execution times for both models. Notably, the GARCH model exhibits relatively swifter performance when applied to smaller data subsets. However, this trend undergoes a complete reversal when larger datasets are considered. Analysing the timing outcomes for the concise window analysis, as presented in Table 5.11, it becomes evident that the GARCH model showcases a speed advantage of 34% in bear markets and 47% in bull markets. On the contrary, delving into the results of the prolonged window evaluation, detailed in Table 5.12, reveals that the LSTM model exhibits superior speed in both bear and bull market conditions, achieving efficiency gains of 66% and 100%, respectively. This analysis underscores the nuanced interplay between model performance and dataset scale, which has implications for selecting the most suitable model for specific use cases.

Table 5.11: Model accuracy - GARCH vs MDL – Long time window.

	Execution Time (Sec)			
	Bear Market		Bull Market	
Prediction (Days)	GARCH	LSTM	GARCH	LSTM
1	33.88	19.90	70.32	33.50
5	32.10	19.73	69.32	33.11
10	31.19	19.81	69.30	35.00
20	34.43	20.07	70.05	37.55

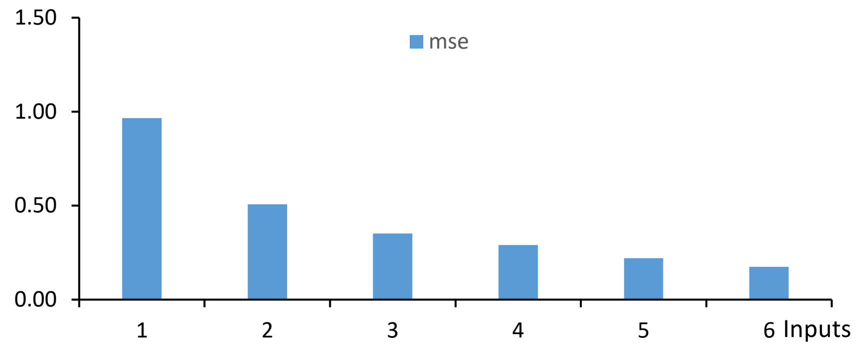
Table 5.12: Model accuracy - GARCH vs MDL – Short time window.

	Execution Time (Sec)			
	Bear Market		Bull Market	
Prediction (Days)	GARCH	LSTM	GARCH	LSTM
1	5.56	8.16	4.58	8.64
5	5.17	8.57	4.60	8.63
10	5.77	8.58	4.85	8.74
20	5.92	8.77	4.54	8.90

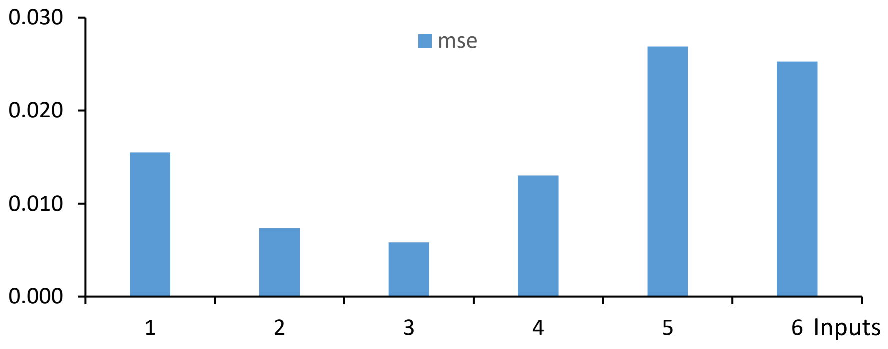
5.3.2 Comparing Multivariate and Single-Input Approaches

Upon the initial examination of the outcomes derived from this series of experiments, a clear pattern emerges. Multivariate Deep Learning (MDL) consistently outperforms the Single-Input Deep Learning (SDL) model across all future prediction horizons and market scenarios. It is important to note, though, that these compelling results are contingent on the varying combinations of inputs deployed to fuel the MDL model.

Figure 5.10 offers a succinct overview of the experiments carried out in this particular set of tests. It illuminates a noteworthy finding: MDL exhibits impressive performance in bear markets across all future prediction durations. However, a distinct trend emerges in bull markets, where models utilizing more than three inputs tend to exhibit underperformance, notably in the 10-day and 20-day future prediction scenarios.



Bear Market.



Bull Market.

Figure 5.10: MTL vs STL – Bear and bull markets.

Following a thorough analysis of the results within bear market conditions, as meticulously presented in Table 5.13, a discernible pattern becomes evident. Most notably, the Multivariate Deep Learning (MDL) model consistently demonstrates its superior performance across all future prediction horizons. Furthermore, an intriguing observation comes to light: an increase in the number of inputs seems to correlate with improved prediction accuracy. This phenomenon underscores the beneficial effect of incorporating a broader array of inputs within the MDL model, ultimately resulting in a substantial enhancement of predictive capabilities, particularly when confronted with challenging market conditions.

Table 5.13: LSTM forecast accuracy of BAC daily volatility in bear market.

	Predictions (Days)			
No. Inputs	1	5	10	20
1	0.7997	1.0563	1.0540	0.9568
2	0.3243	0.6092	0.6426	0.4538
3	0.1971	0.3535	0.4517	0.4060
4	0.1854	0.3106	0.3408	0.3269
5	0.1129	0.2733	0.2906	0.2077
6	0.0833	0.1959	0.1811	0.2354

The summarized results from the tests conducted in the bull market scenario, which serve as the basis for the subsequent analysis, are visually presented in Table 5.14.

Table 5.14: LSTM forecast accuracy of BAC daily volatility in bull market.

	Predictions (Days)			
No. Inputs	1	5	10	20
1	0.0051	0.0167	0.0322	0.0079
2	0.0018	0.0140	0.0063	0.0075
3	0.0010	0.0072	0.0050	0.0100
4	0.0062	0.0162	0.0090	0.0207
5	0.0158	0.0243	0.0342	0.0333
6	0.0097	0.0195	0.0334	0.0385

Conversely, within bull market conditions, the MDL model once again asserts its superiority over SDL. Intriguingly, it worth noting that the most effective MDL configuration was not necessarily the one featuring the highest number of inputs, instead, it was the one with the strongest negative correlation which was for the inputs IBM and GOLD. A full list of correlation between all inputs and BAC is illustrated in Table 5.15. This observation underscores a nuanced trend: while MDL consistently maintains its superiority, the performance dynamics of various input combinations within the MDL framework exhibit divergent behaviours under distinct market conditions. This highlights the intricate interplay between input composition and market context, underscoring the necessity of thoughtful input selection for optimal performance across different scenarios.

Table 5.15: Inputs correlation with BAC in bear and bull markets.

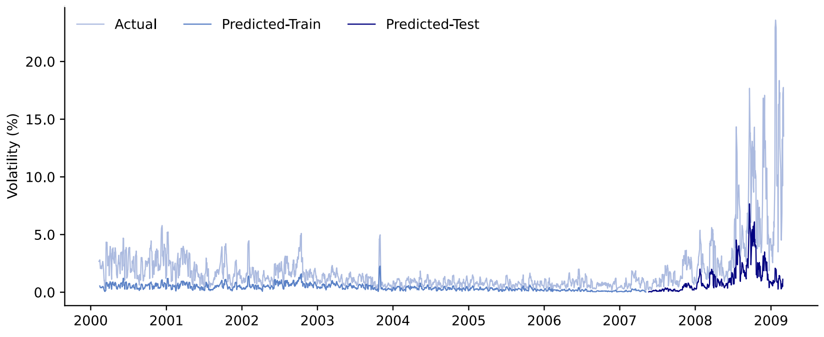
Symbol	Bear Market	Bull Market
JPM	0.53	-0.12
C	0.39	0.72
MS	0.38	0.66
GS	0.72	0.13
DB	0.78	0.73
IBM	-0.09	-0.58
WMT	-0.29	-0.43
GOLD	0.48	-0.53
OIL	0.56	-0.12

Figure 5.11 serves as compelling visual evidence to the unmistakable superiority of Multivariate Deep Learning (MDL) over Single-Input Deep Learning (SDL). This distinction becomes vividly apparent during the training phase, where MDL exhibits an exceptional ability to grasp underlying data trends right from the outset. It steadfastly mirrors the evolving patterns of volatility throughout the learning process, resulting in substantially more robust predictions. Notably, this proficiency persists even in scenarios characterized by a scarcity of data points and the deliberate imposition of challenging hyperparameters.

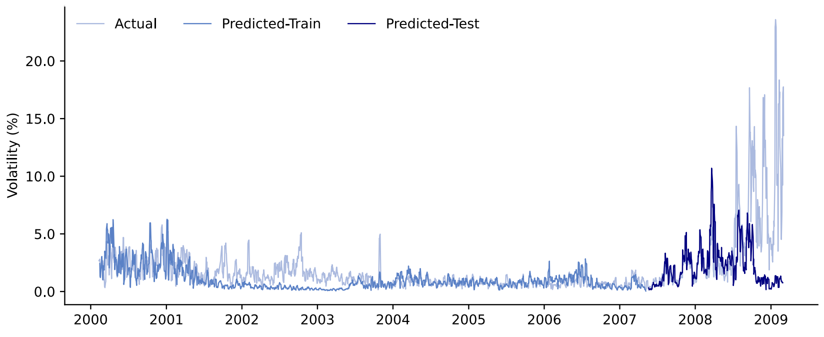
Figure 5.12 exhibits a similar pattern; however, as detailed in Table 5.14, an interesting observation arises. SDL displays a comparatively optimised performance in bull markets than in bear markets. This discrepancy could be attributed to the relative stability of bull markets, where forecasting becomes more straightforward. This distinction is particularly evident in the early stages of the models' learning processes. Nonetheless, it is crucial to emphasize that MDL maintains its position as the superior model.

In conclusion, it is evident that both models encounter difficulties in achieving precise testing predictions. This outcome is a deliberate consequence of the test parameters and data points chosen to facilitate this subjective comparison. It worth noting that an enhanced MDL model, fine-tuned with improved hyperparameters, was deployed in the benchmark testing outlined in Section 5.4.1, with the corresponding results illustrated in Figure 5.6 and Figure 5.7.

The complete results of all the tests conducted in this set have been organized into separate tables, each corresponding to a specific future prediction horizon. These tables can be found in Table 5.16, Table 5.17, Table 5.18, and Table 5.19.

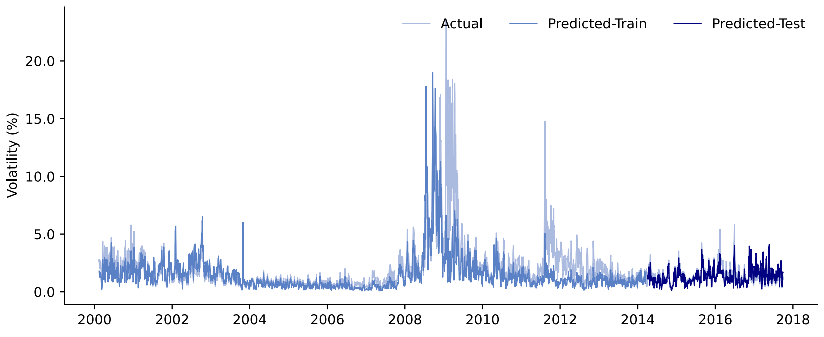


LSTM- SDL model.

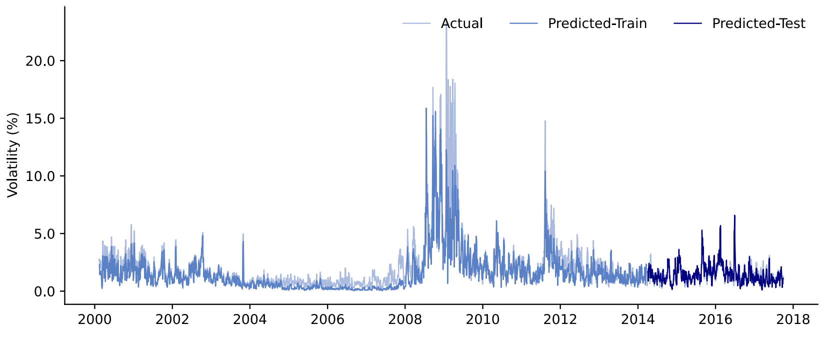


LSTM- MDL model.

Figure 5.11: MDL vs SDL - Volatility prediction for BAC in bear market.



LSTM – SDL model.



LSTM – MDL model.

Figure 5.12: MDL vs SDL - Volatility prediction for BAC in bull market.

Table 5.16: MDL vs SDL 1-day pred. accuracy for BAC daily volatility in bull market.

	Bear Market				Bull Market			
	MSE		SD		MSE		SD	
Input	VAL	DIFF	VAL	DIFF	VAL	DIFF	VAL	DIFF
BAC	0.800	0	0.136	0	0.005	0	0.002	0
BAC, C	0.897	12%	0.127	-7%	0.008	55%	0.008	399%
BAC, DB	0.324	-59%	0.132	-3%	0.017	238%	0.023	1266%
BAC, GOLD	1.442	80%	0.223	64%	0.003	-44%	0.003	93%
BAC, GS	0.416	-48%	0.117	-14%	0.005	0%	0.006	283%
BAC, IBM	0.875	9%	0.280	105%	0.046	795%	0.047	2684%
BAC, JPM	0.405	-49%	0.177	30%	0.015	194%	0.014	719%
BAC, MS	0.457	-43%	0.160	17%	0.003	-46%	0.003	62%
BAC, OIL	1.085	36%	0.433	218%	0.002	-66%	0.003	64%
BAC, WMT	0.582	-27%	0.173	27%	0.043	737%	0.039	2216%
BAC, DB, C	0.376	-53%	0.121	-11%	0.002	-61%	0.001	-25%
BAC, DB, MS	0.293	-63%	0.103	-24%	0.005	-1%	0.005	170%
BAC, GOLD, DB	1.273	59%	0.170	25%	0.018	256%	0.020	1090%
BAC, GOLD, IBM	1.439	80%	0.273	100%	0.001	-80%	0.001	-41%
BAC, GOLD, WMT	1.325	66%	0.277	103%	0.013	161%	0.008	374%
BAC, GS, C	0.286	-64%	0.090	-34%	0.008	58%	0.008	348%
BAC, GS, MS	0.465	-42%	0.107	-21%	0.004	-17%	0.003	72%
BAC, JPM, C	0.662	-17%	0.134	-2%	0.011	118%	0.013	670%
BAC, JPM, DB	0.197	-75%	0.034	-75%	0.020	293%	0.013	652%
BAC, JPM, GS	0.304	-62%	0.088	-35%	0.008	49%	0.007	302%
BAC, JPM, MS	0.368	-54%	0.119	-12%	0.012	124%	0.020	1065%
BAC, MS, C	0.847	6%	0.178	31%	0.001	-74%	0.001	-40%
BAC, OIL, DB	0.692	-13%	0.153	13%	0.003	-45%	0.002	46%
BAC, OIL, IBM	1.036	30%	0.324	137%	0.006	13%	0.007	292%
BAC, OIL, JPM	1.105	38%	0.286	109%	0.009	83%	0.009	411%
BAC, OIL, WMT	0.603	-25%	0.188	38%	0.008	48%	0.004	156%
BAC, DB, GS, C	0.210	-74%	0.047	-65%	0.006	20%	0.008	366%
BAC, DB, GS, MS	0.377	-53%	0.042	-69%	0.013	155%	0.023	1248%
BAC, GS, MS, C	0.200	-75%	0.143	5%	0.007	29%	0.004	135%
BAC, JPM, GS, C	0.185	-77%	0.073	-46%	0.008	48%	0.006	261%
BAC, JPM, GS, DB	0.187	-77%	0.076	-44%	0.014	180%	0.019	1001%
BAC, JPM, GS, MS	0.490	-39%	0.118	-13%	0.019	272%	0.015	791%
BAC, JPM, GS, DB, C	0.113	-86%	0.090	-34%	0.021	308%	0.015	801%
BAC, JPM, GS, DB, MS	0.360	-55%	0.138	1%	0.016	208%	0.009	426%
BAC, JPM, GS, DB, C, MS	0.083	-90%	0.020	-85%	0.010	88%	0.007	292%

Table 5.17: MDL vs SDL 5-day pred. accuracy for BAC daily volatility in bull market.

	Bear Market				Bull Market			
	MSE		SD		MSE		SD	
Input	VAL	DIFF	VAL	DIFF	VAL	DIFF	VAL	DIFF
BAC	1.056	0	0.248	0	0.017	0	0.020	0
BAC, C	0.951	-10%	0.128	-49%	0.039	130%	0.022	11%
BAC, DB	0.609	-42%	0.263	6%	0.061	266%	0.050	149%
BAC, GOLD	1.355	28%	0.433	74%	0.019	11%	0.015	-27%
BAC, GS	0.668	-37%	0.219	-12%	0.024	42%	0.041	104%
BAC, IBM	1.153	9%	0.181	-27%	0.068	308%	0.064	221%
BAC, JPM	0.712	-33%	0.266	7%	0.034	105%	0.037	84%
BAC, MS	0.780	-26%	0.298	20%	0.015	-9%	0.017	-13%
BAC, OIL	0.976	-8%	0.442	78%	0.014	-16%	0.023	14%
BAC, WMT	0.685	-35%	0.154	-38%	0.102	508%	0.077	285%
BAC, DB, C	0.534	-49%	0.237	-5%	0.042	149%	0.047	133%
BAC, DB, MS	0.468	-56%	0.133	-46%	0.007	-56%	0.005	-75%
BAC, GOLD, DB	1.155	9%	0.312	26%	0.036	114%	0.013	-33%
BAC, GOLD, IBM	1.639	55%	0.408	64%	0.007	-57%	0.009	-57%
BAC, GOLD, WMT	1.205	14%	0.284	15%	0.017	4%	0.010	-48%
BAC, GS, C	0.545	-48%	0.391	58%	0.020	21%	0.011	-45%
BAC, GS, MS	0.906	-14%	0.306	23%	0.011	-37%	0.010	-50%
BAC, JPM, C	0.712	-33%	0.104	-58%	0.044	161%	0.039	94%
BAC, JPM, DB	0.353	-67%	0.177	-29%	0.082	391%	0.087	333%
BAC, JPM, GS	0.600	-43%	0.221	-11%	0.043	157%	0.046	131%
BAC, JPM, MS	0.509	-52%	0.136	-45%	0.021	25%	0.028	38%
BAC, MS, C	1.087	3%	0.227	-9%	0.015	-9%	0.037	82%
BAC, OIL, DB	0.819	-22%	0.309	25%	0.026	58%	0.030	50%
BAC, OIL, IBM	1.030	-2%	0.459	85%	0.051	206%	0.029	45%
BAC, OIL, JPM	1.170	11%	0.608	145%	0.027	62%	0.038	88%
BAC, OIL, WMT	0.847	-20%	0.270	9%	0.014	-14%	0.011	-47%
BAC, DB, GS, C	0.353	-67%	0.159	-36%	0.035	109%	0.027	33%
BAC, DB, GS, MS	0.580	-45%	0.353	42%	0.023	38%	0.027	34%
BAC, GS, MS, C	0.540	-49%	0.317	28%	0.017	-1%	0.011	-47%
BAC, JPM, GS, C	0.311	-71%	0.219	-12%	0.016	-3%	0.019	-8%
BAC, JPM, GS, DB	0.539	-49%	0.311	25%	0.022	33%	0.014	-32%
BAC, JPM, GS, MS	0.591	-44%	0.109	-56%	0.026	53%	0.020	-2%
BAC, JPM, GS, DB, C	0.273	-74%	0.094	-62%	0.024	45%	0.011	-43%
BAC, JPM, GS, DB, MS	0.471	-55%	0.131	-47%	0.028	70%	0.013	-34%
BAC, JPM, GS, DB, C, MS	0.196	-81%	0.089	-64%	0.020	17%	0.014	-31%

Table 5.18: MDL vs SDL 10-day pred. accuracy for BAC daily volatility in bull market.

	Bear Market				Bull Market			
	MSE		SD		MSE		SD	
Input	VAL	DIFF	VAL	DIFF	VAL	DIFF	VAL	DIFF
BAC	1.054	0	0.171	0	0.032	0	0.027	0
BAC, C	1.094	4%	0.412	141%	0.056	73%	0.031	17%
BAC, DB	0.643	-39%	0.319	87%	0.066	106%	0.042	59%
BAC, GOLD	1.348	28%	0.399	134%	0.020	-38%	0.018	-32%
BAC, GS	0.759	-28%	0.271	58%	0.049	52%	0.055	108%
BAC, IBM	1.204	14%	0.276	61%	0.085	165%	0.052	95%
BAC, JPM	0.688	-35%	0.299	75%	0.062	92%	0.032	21%
BAC, MS	0.797	-24%	0.284	66%	0.007	-77%	0.005	-81%
BAC, OIL	1.064	1%	0.304	78%	0.006	-80%	0.007	-74%
BAC, WMT	0.928	-12%	0.137	-20%	0.123	282%	0.067	154%
BAC, DB, C	0.599	-43%	0.235	37%	0.052	61%	0.044	65%
BAC, DB, MS	0.653	-38%	0.336	96%	0.031	-5%	0.018	-32%
BAC, GOLD, DB	1.267	20%	0.334	96%	0.086	167%	0.040	52%
BAC, GOLD, IBM	1.497	42%	0.501	193%	0.044	37%	0.044	64%
BAC, GOLD, WMT	1.104	5%	0.248	45%	0.040	24%	0.028	3%
BAC, GS, C	0.538	-49%	0.254	49%	0.017	-46%	0.015	-42%
BAC, GS, MS	0.613	-42%	0.406	138%	0.023	-29%	0.015	-45%
BAC, JPM, C	0.858	-19%	0.166	-3%	0.049	52%	0.042	57%
BAC, JPM, DB	0.486	-54%	0.290	69%	0.074	130%	0.046	74%
BAC, JPM, GS	0.751	-29%	0.380	122%	0.081	151%	0.127	377%
BAC, JPM, MS	0.728	-31%	0.415	142%	0.024	-26%	0.045	71%
BAC, MS, C	1.008	-4%	0.253	48%	0.005	-84%	0.004	-83%
BAC, OIL, DB	0.794	-25%	0.278	62%	0.056	75%	0.044	66%
BAC, OIL, IBM	1.034	-2%	0.366	114%	0.062	91%	0.057	112%
BAC, OIL, JPM	1.343	27%	0.588	244%	0.039	21%	0.031	15%
BAC, OIL, WMT	0.755	-28%	0.241	41%	0.047	45%	0.039	46%
BAC, DB, GS, C	0.380	-64%	0.235	37%	0.029	-9%	0.034	29%
BAC, DB, GS, MS	0.508	-52%	0.074	-57%	0.068	110%	0.060	125%
BAC, GS, MS, C	0.401	-62%	0.138	-19%	0.013	-61%	0.011	-59%
BAC, JPM, GS, C	0.341	-68%	0.167	-2%	0.009	-72%	0.005	-83%
BAC, JPM, GS, DB	0.690	-34%	0.444	160%	0.039	21%	0.036	37%
BAC, JPM, GS, MS	0.824	-22%	0.363	112%	0.032	-1%	0.020	-26%
BAC, JPM, GS, DB, C	0.291	-72%	0.207	21%	0.034	6%	0.024	-9%
BAC, JPM, GS, DB, MS	0.517	-51%	0.161	-6%	0.043	35%	0.019	-28%
BAC, JPM, GS, DB, C, MS	0.181	-83%	0.111	-35%	0.033	4%	0.018	-33%

Table 5.19: MDL vs SDL 20-day pred. accuracy for BAC daily volatility in bull market.

	Bear Market				Bull Market			
	MSE		SD		MSE		SD	
Input	VAL	DIFF	VAL	DIFF	VAL	DIFF	VAL	DIFF
BAC	0.957	0	0.148	0	0.008	0	0.011	0
BAC, C	0.994	4%	0.199	35%	0.062	678%	0.028	158%
BAC, DB	0.454	-53%	0.259	76%	0.070	778%	0.052	373%
BAC, GOLD	1.132	18%	0.325	120%	0.025	218%	0.030	176%
BAC, GS	0.585	-39%	0.226	53%	0.054	578%	0.075	591%
BAC, IBM	1.142	19%	0.097	-34%	0.149	1779%	0.055	399%
BAC, JPM	0.680	-29%	0.242	64%	0.059	641%	0.019	70%
BAC, MS	0.679	-29%	0.261	77%	0.011	38%	0.007	-32%
BAC, OIL	0.900	-6%	0.197	34%	0.007	-6%	0.004	-66%
BAC, WMT	0.770	-20%	0.203	37%	0.137	1628%	0.159	1357%
BAC, DB, C	0.567	-41%	0.329	123%	0.106	1238%	0.101	828%
BAC, DB, MS	0.484	-49%	0.308	109%	0.038	375%	0.037	236%
BAC, GOLD, DB	1.114	16%	0.176	19%	0.104	1216%	0.062	463%
BAC, GOLD, IBM	1.026	7%	0.197	33%	0.032	307%	0.031	180%
BAC, GOLD, WMT	0.979	2%	0.273	85%	0.047	495%	0.037	242%
BAC, GS, C	0.443	-54%	0.132	-10%	0.024	209%	0.014	30%
BAC, GS, MS	0.495	-48%	0.116	-22%	0.023	186%	0.009	-13%
BAC, JPM, C	0.785	-18%	0.184	24%	0.081	929%	0.058	428%
BAC, JPM, DB	0.492	-49%	0.226	53%	0.077	872%	0.075	583%
BAC, JPM, GS	0.406	-58%	0.115	-22%	0.038	381%	0.037	236%
BAC, JPM, MS	0.510	-47%	0.270	83%	0.031	298%	0.031	179%
BAC, MS, C	1.017	6%	0.211	43%	0.010	27%	0.014	31%
BAC, OIL, DB	1.189	24%	0.563	282%	0.069	769%	0.030	174%
BAC, OIL, IBM	0.917	-4%	0.376	155%	0.067	743%	0.042	282%
BAC, OIL, JPM	1.148	20%	0.513	248%	0.025	217%	0.035	219%
BAC, OIL, WMT	0.694	-27%	0.199	35%	0.043	437%	0.020	85%
BAC, DB, GS, C	0.388	-59%	0.421	186%	0.023	185%	0.017	54%
BAC, DB, GS, MS	0.542	-43%	0.298	102%	0.052	559%	0.038	243%
BAC, GS, MS, C	0.721	-25%	0.280	90%	0.021	162%	0.009	-13%
BAC, JPM, GS, C	0.327	-66%	0.106	-28%	0.038	380%	0.043	296%
BAC, JPM, GS, DB	0.367	-62%	0.113	-23%	0.044	453%	0.027	146%
BAC, JPM, GS, MS	0.609	-36%	0.237	61%	0.042	435%	0.018	64%
BAC, JPM, GS, DB, C	0.208	-78%	0.073	-50%	0.033	320%	0.017	55%
BAC, JPM, GS, DB, MS	0.530	-45%	0.149	1%	0.038	384%	0.015	38%
BAC, JPM, GS, DB, C, MS	0.235	-75%	0.222	51%	0.038	387%	0.016	50%

5.4 Conclusions

The primary objectives of this study encompass addressing the viability of deploying Long Short-Term Memory (LSTM) as an alternative to conventional statistical models such as Generalized Autoregressive Conditional Heteroskedasticity (GARCH). This investigation seeks to assess LSTM potential for predicting market volatility, constructing more profitable trading portfolios, and enhancing options pricing models.

The secondary objective revolves around the proposal and examination of a multivariate LSTM architecture with the aim of elevating the accuracy of volatility forecasting. Benchmark testing results validate the initial hypothesis, unequivocally demonstrating that deep learning, specifically LSTM-based volatility forecasting, outperforms GARCH. This outcome further underscores LSTM suitability for modelling intra-day volatility, particularly owing to its proficiency in detecting short-term market influencers.

Subsequent test sets reveal that a multivariate LSTM can notably enhance volatility forecasting in both bear and bull markets. Notably, improvements are evident in both 1-day and 5-day predictions when incorporating additional asset prices into the multivariate input, corroborating the second hypothesis. Additionally, it becomes apparent that introducing a lag window is essential for refining LSTM prediction accuracy, especially in bear market conditions. This lag window aids in training the model to adapt to rapid market fluctuations and accommodate potential gaps in data.

Chapter 6

Multitasking in Trading Markets: Enhancing Decision-Making with Multi-Task Deep Learning

The stock market, which is referred to as the equity market, is a fundamental component of the global financial system where shares of publicly traded companies are bought and sold. It functions as a marketplace where investors, such as individuals and institutions, can purchase ownership shares in companies. These shares represent a portion of the company ownership and entitle the shareholder to a portion of its profits and potential growth [13].

A trading portfolio, commonly known as an investment portfolio or securities portfolio, is a collection of various financial assets held by an individual or entity for the purpose of trading or investment. It encompasses a diverse range of securities, such as stocks, indices, bonds, mutual funds, exchange-traded funds, options, and other investment instruments [10], [17].

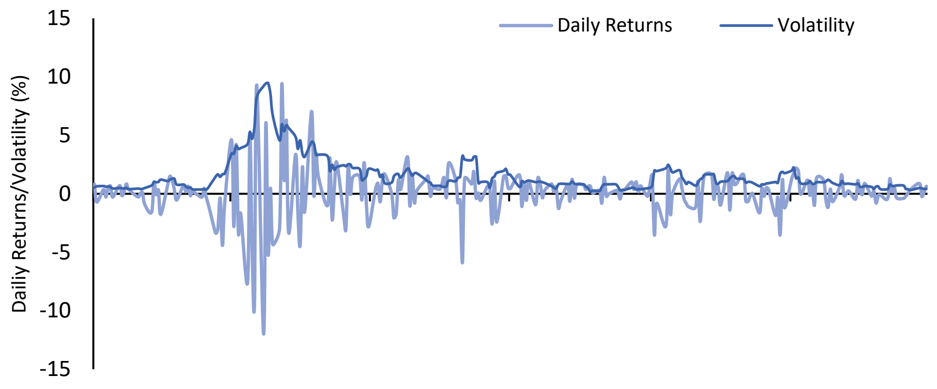
Effective risk management is crucial in portfolio trading. Investors analyse and assess the risk associated with each investment, considering factors such as market volatility, economic conditions, and company performance.

Investors seek to generate returns on their trading portfolio through capital appreciation, increase in asset value and income from dividends, interest, etc.

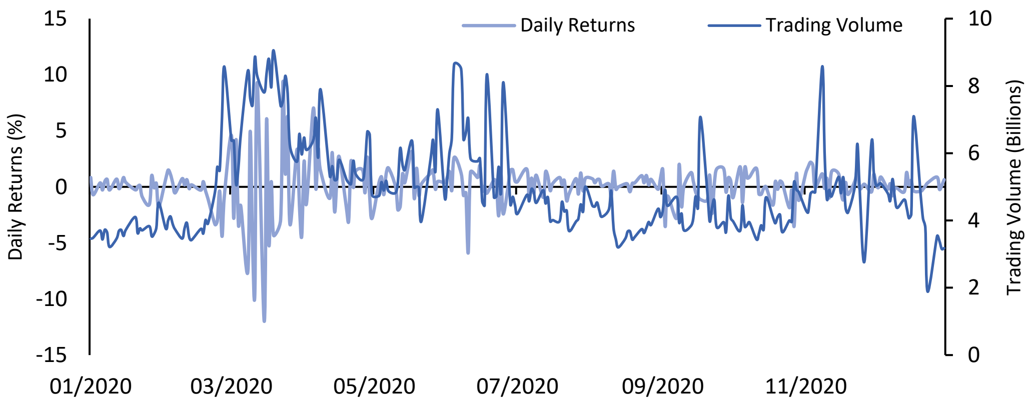
Portfolios can be actively managed, where investment decisions are actively made based on market analysis, or passively managed, where investments track a specific market index or benchmark [10].

The primary objective of a trading portfolio is to achieve specific financial goals, which can include capital appreciation, income generation, risk mitigation, and wealth preservation. Traders and investors strategically allocate their funds across different assets to achieve a balance between risk and return, aiming to optimize their overall portfolio performance [13].

Vigilant monitoring of stock market performance is a critical factor in achieving successful investments. By formulating suitable trading strategies, pricing models, and risk management tools, investors and financial firms can continuously assess and react to market fluctuations, strategically positioning themselves to capitalize on profitable opportunities. This approach enhances their chances of being on the winning side of trades and optimizing overall portfolio performance [10], [13].



S&P 500 Daily returns and volatility.



S&P 500 daily returns and trading volumes.

Figure 6.1: S&P 500 volatility, daily returns, and trading volume - COVID-19 (2020).

Volatility, returns, and trading volumes are fundamental metrics extensively utilized in the stock market performance monitoring, with volatility also recognized as a critical Key Performance Indicator (KPI) in this domain [9]. Multiple studies have presented evidence of correlations between these metrics, including the relationship between volatility and returns, volatility and trading volume, and returns and trading volume [71], [72], [73], [74]. Figure 6.1

depicts the correlation among volatility, returns, and trading volumes for the SP500 index during COVID-19 pandemic peak in 2020.

Effectively monitoring stock market performance is paramount for achieving successful investments. Investors and financial firms can enhance their chances of success by developing suitable trading strategies, pricing models, and risk management tools, enabling them to continuously monitor and respond to market changes and position themselves on the winning side of trades [75]. This proactive approach helps produce superior returns while controlling risk, contributing to overall portfolio optimization [10].

Over the course of several decades, both practitioners and academics have extensively researched the forecasting of volatility. Improved prediction accuracy in market volatility plays a crucial role in enhancing risk management and pricing models, leading to more effective trading and investment strategies [76].

Currently, statistical models such as GARCH and ARIMA are commonly deployed in financial applications to forecast volatility and stock market price movements. However, with the rapid increase in trading volumes and market complexity, driven by faster Internet technologies and mobile cellular networks, there is a growing demand for alternative methods that can process vast amounts of data with higher speed and accuracy.

Artificial Neural Networks (ANN) and Deep Learning have emerged as promising approaches due to their algorithmic improvements, the availability of large datasets, and advancements in computing hardware, such as TensorFlow cores, which enable the processing of extensive data with these specific algorithms [7], [8], [41], [77].

The aim of this research to address the third hypothesis of the thesis which is the use of multi-task deep learning model to improve prediction accuracy of key stock market metrics by taking advantage of common shared layer of all tasks and their distributed loss function.

In this research, a Multi-Task Deep Learning (MTDL) Recurrent Neural Network (RNN) model was constructed, incorporating various Deep Learning (DL) algorithms, namely Long Short-Term Memory (LSTM), Gated Recurrent Unit (GRU), and Recurrent Neural Network (RNN). The primary focus of the research was to predict Volatility, while also considering daily returns and trading volume as secondary targets. To optimize the model performance, a shared layer was utilized for all input data, and a dynamic loss function was introduced, enabling for the tuning of weights favouring specific outputs.

The results of all different DL algorithms were compared with GARCH, a statistical model used for Volatility and Trading Volume modelling, and ARIMA, another statistical model applied for index price prediction. To provide a comprehensive analysis, the same set of tests were replicated on a Single-Task Deep Learning (STDL) model and compared the outcomes with those obtained from the Multi-Task Deep Learning (MTDL) model.

The upcoming portions of this chapter are organized in the subsequent manner:

Section 6.1: In Section 6.1, an extensive review of literature is presented, examining the realm of multi-task deep learning both in a broader context and, more specifically, in the finance domain concerning the prediction of crucial stock market metrics.

Section 6.2: Section 6.2 provides an in-depth explanation of the methodology chosen for constructing the multi-task deep learning and statistical models. This section investigates thoroughly the testing protocols, variables utilized, intricacies of the dataset, and the range of experiments carried out.

Section 6.3: The findings obtained from benchmark tests are detailed in Section 6.3, accompanied by a comparative assessment of single-task and multi-task deep learning models.

6.1 Background and Related Works

In the context of trading markets, multitasking refers to the strategic utilization of various analytical tasks simultaneously to enhance decision-making processes. By concurrently considering multiple dimensions of market behaviour and related factors, traders aim to gain a more comprehensive and nuanced understanding of market dynamics. This approach enables for more informed and potentially more effective trading strategies [53].

Multitasking involves processing and analysing multiple streams of data concurrently. This includes examining historical price trends, trading volumes, market news, sentiment analysis, and various technical indicators to capture a holistic view of the market environment.

Different analytical tasks often provide complementary insights. For instance, while technical analysis may highlight potential price patterns, fundamental analysis can shed light on the underlying factors influencing those patterns. Combining these insights can lead to more robust trading decisions.

Multitasking extends to risk assessment and management. Traders simultaneously evaluate diverse risk factors, such as market volatility, potential economic events, and geopolitical developments, to make well-informed decisions that mitigate potential losses [54].

Markets are dynamic and can change rapidly, multitasking enables traders to monitor various indicators in real-time, facilitating prompt responses to evolving market conditions [52].

There is a noticeable gap in the existing literature when it comes to investigating the application of multitasking methodologies within the context of the stock market. Despite the significant advancements in deep learning and its various applications in financial analysis, there remains a limited examination of multitasking approaches specifically tailored to stock market prediction and portfolio management.

While numerous studies have investigated deep learning techniques for stock market prediction and risk assessment, the integration of multitasking principles appears to be an unexamined avenue. Multitasking has shown promise in other domains, such as natural language processing and computer vision, yet its potential to enhance predictive accuracy and decision-making in the stock market realm remains relatively uncharted.

By addressing this gap in the literature, the current research seeks to contribute to a more comprehensive understanding of how multitasking strategies can be harnessed to improve the prediction of stock market behaviour, portfolio management, and risk assessment. Through a meticulous study of multitasking models, loss functions, and their interplay within the financial domain, this study aims to shed light on the untapped potential of multitasking deep learning in driving innovation and precision within stock market analysis and decision-making.

6.2 Methodology

A series of experiments were conducted to assess the effectiveness of multi-task learning compared to single task learning and traditional statistical models. These experiments involved the implementation of various deep learning architectures and statistical models to evaluate their accuracy in predicting different stock market metrics. These assessments are also essential for the validation and examination of “Hypothesis 3” and its associated sub-hypotheses.

6.2.1 Testing Modules and Metrics

The research involved multiple experiments focused on different stock market metrics, utilizing statistical and deep learning models. The tests are executed for different market conditions, different future prediction horizons, and different combinations of input tasks. Mean Squared Error (MSE) is used as the performance metrics between actual predicted metric and the predicted one.

Three main test modules were created to achieve this:

GARCH Module: This module takes an input variable representing daily returns, feeds it to a GARCH model, and compares the output (realized volatility) against actual volatility. The Mean Squared Error (MSE) is used to measure the model accuracy.

ARIMA Module: This module takes an input variable representing any stock market metric, feeds it to an ARIMA model, and compares the output against actual data. MSE is used to measure the model accuracy.

Single-Task Deep Learning Module: This module expects multiple input variables representing a single or a collection of stock market metrics. It feeds the input to the model, and a single output is compared against actual data. MSE is used to measure the model accuracy.

Multi-Task Deep Learning Module: This module also expects multiple input variables representing a single or a collection of stock market metrics. It feeds the input to the model, and the output, consisting of 2 or more metrics, is compared against actual data. MSE is used to measure the model accuracy.

6.2.2 Data Collection

Daily closing prices for main stock indexes were downloaded using Yahoo Finance Application Programmable Interface (API). Formulas elucidated in Section 2.5, pertaining to Volatility, and Section 2.6, concerning Returns, were deployed to compute both Returns and 5-Day rolling volatility. Full details of the data sets are illustrated in Table 6.1 and Table 6.2.

Table 6.1: Test data definition - time series.

Market	Training Data
Bear	29/08/2007 - 29/01/2009
Bull	27/08/2018 - 29/01/2020
Volatile	28/08/2019 - 29/01/2021

Table 6.2: Test data definition - Indexes.

Symbol	Name	Type
GSPC_PRICE	S&P 500 Index Price	Observed
DJI_PRICE	Dow Jones Index Price	Observed
NYA_PRICE	New York Stock Exchange Index	Observed
GSPC_VOLUME	S&P 500 Index Trading Volume	Observed
DJI_VOLUME	Dow Jones Index Trading Volume	Observed
GSPC_RETURNS	S&P 500 Index Daily Returns	Calculated
DJI_RETURNS	Dow Jones Index Daily Returns	Calculated
GSPC_VOLATILTIY	S&P 500 Index Volatility	Calculated
DJI_VOLATILTIY	Dow Jones Index Volatility	Calculated

6.2.3 Experimental Procedures

During the testing stage, *Python 3.9* and *TensorFlow 2.1* were deployed to create Multi-Task and Single-Task learning RNN architectures. To conduct benchmark assessments, GARCH and ARIMA statistical models were coded using the *arch* and *statsmodels.tsa.arima.model* libraries within *Python 3.9.* All experimentation took place within a uniform testing environment and application context. Sample pseudocode illustrating GARCH and ARIMA models training are included in Appendix D.

In all the trials conducted, the hold-out performance estimation technique was deployed. The training data consisted of historical stock prices, serving as input for both the DL networks and statistical models. These experiments were designed to cover a spectrum of market scenarios, spanning bear, bull, and volatile markets. This diversity enabled us to thoroughly evaluate the models' effectiveness under varying conditions. To ensure the reliability of the findings, each test was repeated 10 times, resulting in a comprehensive set of outcomes. The analysis encompassed not only the average Mean Squared Error (MSE) but also the standard deviation of MSE, providing a comprehensive and in-depth perspective. It worth noting that a total of over 8,000 tests were executed for this test.

The selection of hyperparameters and the overall optimization of the models deployed in the testing phase were guided by the number of parameters present in the newly compared models. You can find a comprehensive account of the optimization process for deep learning models in Appendix B, which provides detailed insights into this aspect of the research.

The details of the server, GPU card, and other computational resources utilized for the entire testing process are listed in Table 6.3 and Table 6.4.

Table 6.3: Test environment – Servers specifications.

Server Specifications	
CPU	Intel Xeon CPU E5-2640 @3.0GHz, 2 processors with 24 cores)
Memory	128 GB
Environment	UBUNTU 22.04
Language	Python 3.9
ML Framework	TensorFlow 2.0

Table 6.4: Test environment - GPU specifications.

GPU Specifications	
GPU	GeForce RTX 2060 SUPER
Memory	8 GB
CUDA Cores	2176
GPU Clock	1650 MHz

6.2.3.1 Evaluation through Benchmark Testing

The intention behind these experiments is to juxtapose the precision of multi-task learning against the benchmark statistical models, GARCH and ARIMA. This comparison is also crucial for substantiating the crux of "*Hypothesis 3*".

The conducted statistical model tests are detailed in Table 6.5. Each test is performed utilizing data from bull, bear, and volatile market conditions. It worth mentioning that GARCH is exclusively deployed for modelling volatility, thus limiting its application to the 5-day rolling projected volatility predictions of the S&P 500 Index and Dow Jones Index.

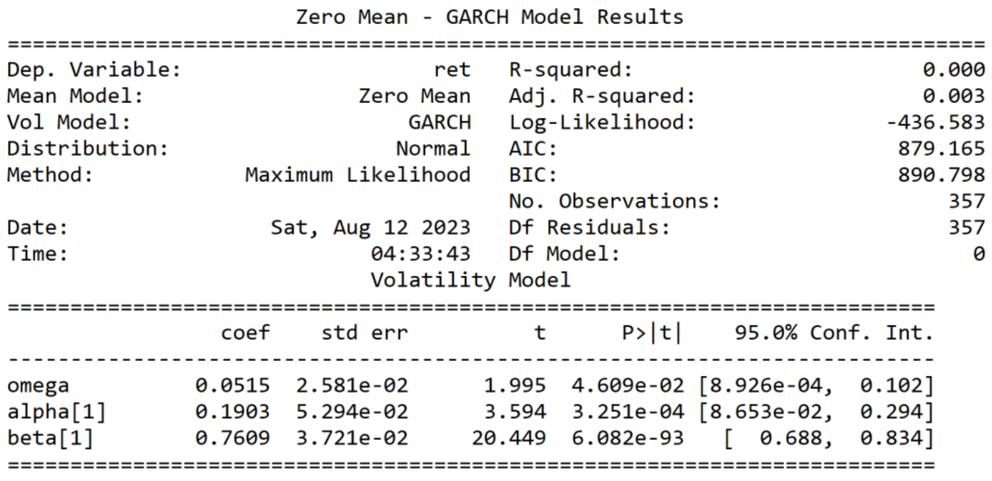
Table 6.5: Benchmark - Input and output for GARCH and ARIMA models tests.

Model	Input/ Output
ARIMA	[GSPC PRIC]
ARIMA	[GSPC RETURNS]
ARIMA, GARCH	[GSPC VOLATILTIY]
ARIMA	[GSPC VOLUME]
ARIMA	[DJI PRIC]
ARIMA	[DJI RETURNS]
ARIMA, GARCH	[DJI VOLATILTIY]
ARIMA	[DJI VOLUME]

Code 6.1: GARCH – Selecting optimal parameters.

```
1: for p from 1 to trials do
2:   for q from 1 to trials do
3:     garch ← model.fit(returns, mean='zero', vol='GARCH', p, o, q)
4:     append garch.bic to bic_garch
5:     if garch.bic equals minimum value of bic_garch then
6:         best_param ← (p, q)
7:     end if
8:   end for
9: end for
```

The GARCH models were constructed by determining the optimal values for "p" and "q," which correspond to the autoregressive (AR) and moving average (MA) orders in the model, respectively. To identify these optimal values, the process outlined in Code 6.1 was deployed. This procedure involves cycling through various combinations of "p" and "q," fitting the model with the cyclic values, and then comparing the results against the smallest value among all prior cycles. When newly derived outcomes prove smaller, they replace the current values; otherwise, they are disregarded. During model training, daily returns were deployed, and the data was scaled using the training dataset. Subsequently, the testing data underwent scaling based on the training data before being used to assess the model. A sample GARCH model summary is showcased in Figure 6.2 which was used to predict S&P 500 5-day rolling realised volatility.



```
                       Zero Mean - GARCH Model Results
==============================================================================
Dep. Variable:                    ret   R-squared:                       0.000
Mean Model:                 Zero Mean   Adj. R-squared:                  0.003
Vol Model:                      GARCH   Log-Likelihood:               -436.583
Distribution:                  Normal   AIC:                           879.165
Method:            Maximum Likelihood   BIC:                           890.798
                                        No. Observations:                  357
Date:                Sat, Aug 12 2023   Df Residuals:                      357
Time:                        04:33:43   Df Model:                            0
                             Volatility Model
==========================================================================
                 coef    std err          t      P>|t|    95.0% Conf. Int.
--------------------------------------------------------------------------
omega          0.0515  2.581e-02      1.995  4.609e-02 [8.926e-04,  0.102]
alpha[1]       0.1903  5.294e-02      3.594  3.251e-04 [8.653e-02,  0.294]
beta[1]        0.7609  3.721e-02     20.449  6.082e-93   [  0.688,  0.834]
==========================================================================
```

Figure 6.2: GARCH model summary.

Code 6.2: ARIMA – Prediction of a single day logic.

```
1: **for** x in range(length of testData) **do**
 2:     model ←ARIMA(history, order=(p, o, q))
 3:     model_fit ← model.fit()
 4:     output ← model_fit.forecast()
 6:     append output[0] to predictions
 8:     nextDay ← testData[x]
 9:     append nextDay to history
10: **end for**
```

ARIMA models were constructed to assess prices, daily returns, volatility, and trading volume. The testing procedure commences by inputting data into the model and determining the optimised "p" and "q" values, following a process akin to the one elucidated for GARCH. Once these values are established, the procedure in Code 6.2 is activated to forecast forthcoming values of the chosen stock market metrics, as enumerated in Table 6.5.

A representative summary of an ARIMA model, along with the model diagnostic results, is portrayed in Figure 6.3 and Figure 6.4, respectively.

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  356
Model:                 ARIMA(1, 1, 0)   Log Likelihood                  81.183
Date:                Sat, 12 Aug 2023   AIC                           -158.366
Time:                        03:18:29   BIC                           -150.622
Sample:                             0   HQIC                          -155.286
                                - 356
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.0190      0.041     -0.462      0.644      -0.100       0.062
sigma2         0.0371      0.002     19.551      0.000       0.033       0.041
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):               122.95
Prob(Q):                              0.97   Prob(JB):                         0.00
Heteroskedasticity (H):               0.47   Skew:                            -0.54
Prob(H) (two-sided):                  0.00   Kurtosis:                         5.67
===================================================================================
```

Figure 6.3: ARIMA model summary – S&P 500 price prediction.

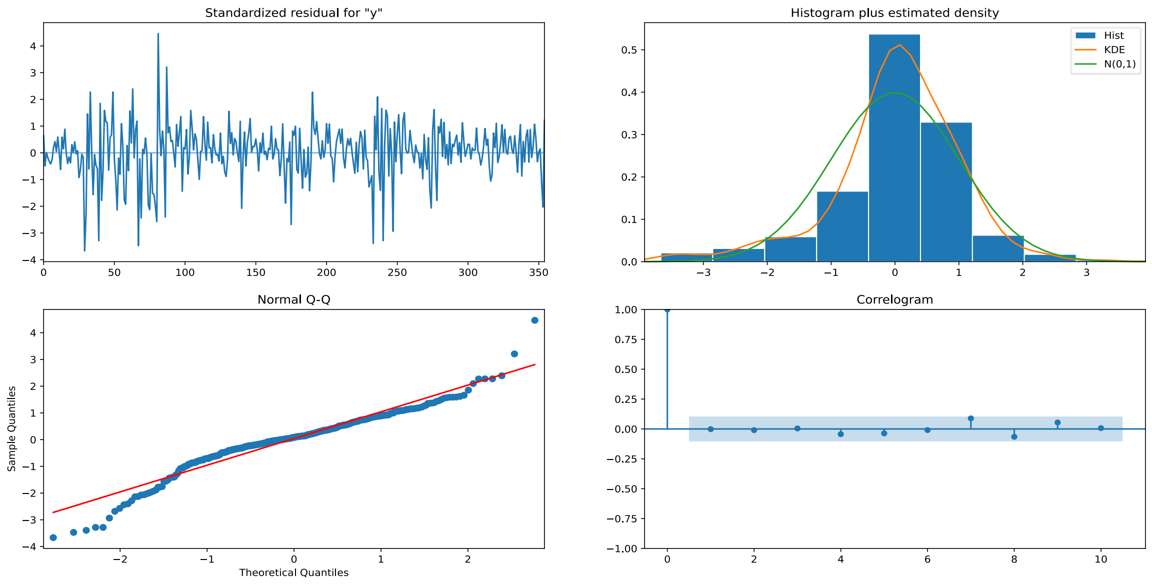


Figure 6.4: ARIMA model diagnostics – S&P 500 price prediction.

Moreover, dual feature multi-task assessments were undertaken for the identical symbols as projected by the statistical models. Each assessment concurrently predicted two tasks. The hyperparameters utilized for the Multi-Task Deep Learning (MTLDL) model can be located within Table 6.6.

Table 6.6: Benchmark - MTDL hyperparameters used in the benchmark tests.

Epochs	Output Neurons	Batch Size	Previous Days	Optimization Algorithm
100	16	100	1	RMSProp

Table 6.7: Benchmark - Input, output, and loss function weights used in MTDL tests.

Model	DL Network	Input/Output	Weights
MTDL	LSTM, GRU, RNN	[GSPC PRICE, DJI PRICE]	[0.5, 0.5]
MTDL	LSTM, GRU, RNN	[GSPC RETURNS, DJI RETURNS]	[0.5, 0.5]
MTDL	LSTM, GRU, RNN	[GSPC VOLATILTIY, DJI VOLATILTIY]	[0.5, 0.5]
MTDL	LSTM, GRU, RNN	[GSPC VOLUME, DJI VOLUME]	[0.5, 0.5]

Similarly to GARCH and ARIMA, the multi-task test application rescales the data before conducting tests. It employs the scaling used for training data on both the training and testing data. Consequently, all predicted values are rescaled before being used to generate charts.

The testing procedure examines both the inputs and weights. If there are multiple inputs, it evaluates the weights. In cases where a single weight is utilized, the model is categorized as a multivariate single-task model. However, when multiple weights are deployed, the application activates the multitasking segment of the model, passing these weights during the training process. The logic used by the testing application is illustrated in Code 6.3.

An exemplar multi-task deep learning model is visualized in Figure 6.5. This particular model was deployed to concurrently forecast prices of both the S&P 500 and Dow Jones indices.

Optimization algorithm, Root Mean Square Propagation (RMSProp) is used for all tests which is used to update the weights on the neural network during training.

Code 6.3: Switching between single-mode and muti-task mode.

```
procedure testModel(inputs, weights):
    **if** count(inputs) > 1:
        evaluateWeights(weights)
    **end if**
    **if** count(inputs) == 1:
        modelType = "multivariate single-task model"
    **end if**
    **if** count(weights) > 1:
        activateMultitasking(weights)
    **end if**

procedure evaluateWeights(weights):
    **if** count(weights) == 1:
        modelType = "multivariate single-task model"
    **end if**
    **if** count(weights) > 1:
        activateMultitasking(weights)
    **end if**

procedure activateMultitasking(weights):
    application.activateMultitaskingSegment(weights)
    modelType = "multitasking model"
    trainModelWithWeights(weights)
    **end if**
```

Ultimately, the datasets utilized for all tests were confined within narrow windows, each reflecting distinct market conditions as outlined in Table 6.2. These condensed datasets were intentionally chosen to exert pressure on all models due to their limited number of data points. This constraint heightens the challenge of achieving precise predictions, thereby facilitating a subjective evaluation of performance across various statistical and deep learning models, as well as between single-task and multi-task deep learning models.

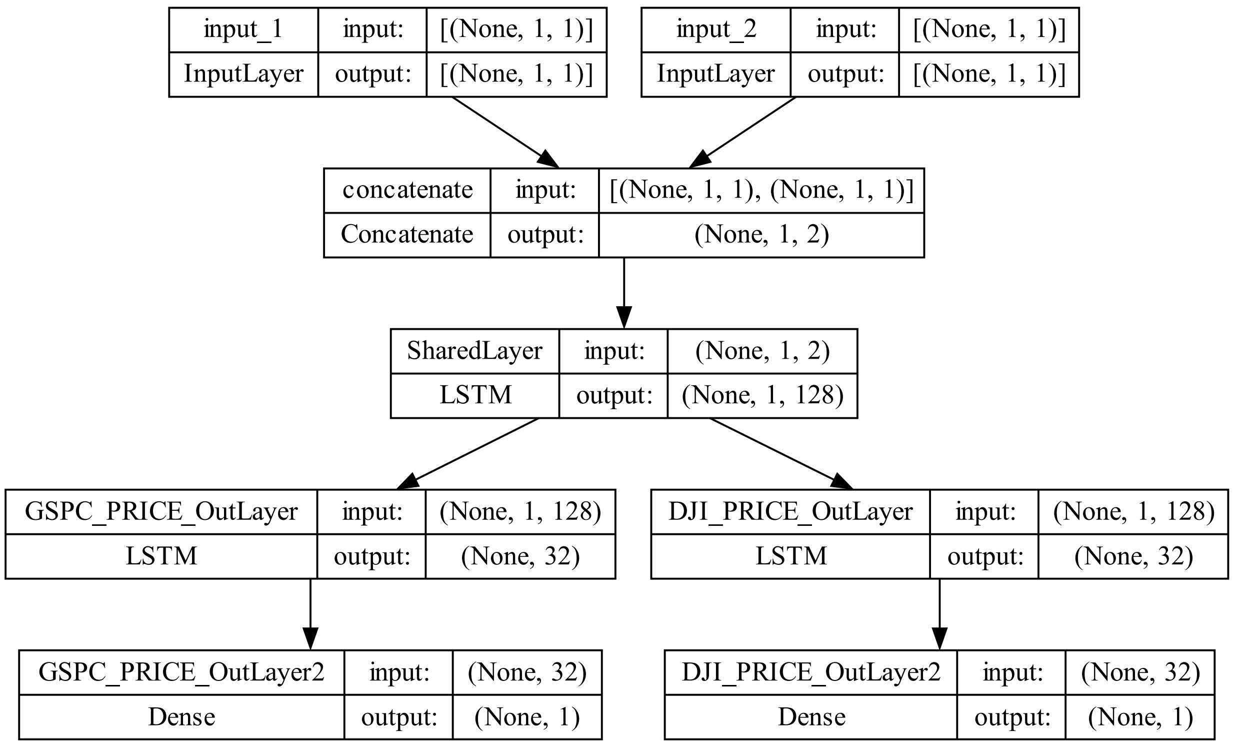

input_1 | InputLayer | input: [(None, 1, 1)] | output: [(None, 1, 1)]
input_2 | InputLayer | input: [(None, 1, 1)] | output: [(None, 1, 1)]
concatenate | Concatenate | input: [(None, 1, 1), (None, 1, 1)] | output: (None, 1, 2)
SharedLayer | LSTM | input: (None, 1, 2) | output: (None, 1, 128)
GSPC_PRICE_OutLayer | LSTM | input: (None, 1, 128) | output: (None, 32)
DJI_PRICE_OutLayer | LSTM | input: (None, 1, 128) | output: (None, 32)
GSPC_PRICE_OutLayer2 | Dense | input: (None, 32) | output: (None, 1)
DJI_PRICE_OutLayer2 | Dense | input: (None, 32) | output: (None, 1)

Figure 6.5: Benchmark – MTDL two-tasks model diagram.

6.2.3.2 Comparing Multitasking and Singl-Task Approaches

This series of meticulously planned experiments serves a dual purpose, meticulously designed to rigorously validate the core thesis of this research and achieve two pivotal objectives. Firstly, its primary aim is to establish the superiority of Multi-Task Deep Learning (MTDL) in contrast to Single-Task Deep Learning (STDL) models, thereby reaffirming the central hypothesis that forms the crux of this study. Secondly, it thoroughly examines and compares the performance of two prominent RNN architectures, namely LSTM and GRU, within the contexts of both MTDL and STDL frameworks.

The comprehensive testing procedures are meticulously documented in Table 6.7, delineating the specific tests conducted for the MTDL framework, and in Table 6.8, which provides an intricate breakdown of the tests carried out within the STDL framework. These tables serve as essential reference points, elucidating the experimental design and the nuances of each test, ensuring transparency and replicability in the pursuit of empirical insights and robust conclusions.

For MTDL tests, the number of neurons used in the shared layer and epochs are multiplied by the number of outputs. For STDL, the hyper parameters used are matching those used for MTDL which are listed in Table 6.6.

Figure 6.6 visually represents one of the conducted tests, where, in this instance, it predicts the prices of the S&P 500 index.

Table 6.8: Input, output, and loss function weights used in STDL tests.

Model	Input/Output	Weights
STDL	[GSPC PRICE]	[1.0]
STDL	[GSPC RETURNS]	[1.0]
STDL	[GSPC VOLATILITY]	[1.0]
STDL	[GSPC VOLUME]	[1.0]
STDL	[DJI PRICE]	[1.0]
STDL	[DJI RETURNS]	[1.0]
STDL	[DJI VOLATILTIY]	[1.0]
STDL	[DJI VOLUME]	[1.0]

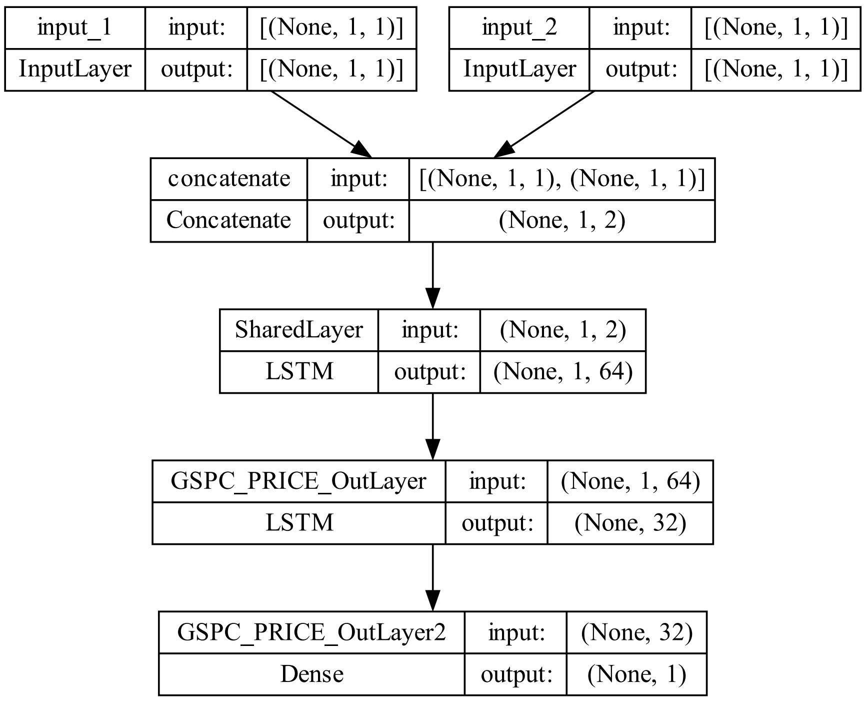


Figure 6.6. MTDL vs STDL – STDL model diagram.

6.2.3.3 The Art of Multitasking – Finding the Optimal Task Combination

The aim of this series of experiments is to confirm the primary research goal along with one of the secondary hypotheses within "Hypothesis 3." This hypothesis posits that refining the accuracy of Multi-Task Deep Learning (MTDL) is achievable by identifying the optimal fusion of interrelated metrics. To accomplish this, a subjective approach is deployed by selecting one poorly predicted MTDL metric from benchmark testing results and using it for this specific test. Tests performed in this set are listed in Table 6.9.

Table 6.9: Input, output, and loss function weights used in different combinations tests.

Model	Input/Output	Weights	Market
MTDL	[GSPC PRICE, DJI RETURNS]	[0.5, 0.5]	Bear
MTDL	[GSPC PRICE, DJI VOLATILITY]	[0.5, 0.5]	Bear
MTDL	[GSPC PRICE, DJI VOLUME]	[0.5, 0.5]	Bear
MTDL	[GSPC PRICE, GSPC VOLATILITY, DJI VOLATILITY]	[0.333, 0.333,0.333]	Bear

6.2.3.4 Fine-Tuning the Loss Function: Determining the Optimal Weights

The purpose of conducting this series of experiments is to verify the primary research objective and one of the subsidiary hypotheses within "*Hypothesis 3*". This particular sub-hypothesis proposes that enhancing the accuracy of Multi-Task Deep Learning (MTDL) can be achieved by ascertaining the optimal weights for the output components. To achieve this, a subjective approach is adopted by selecting one poorly MTDL predicted metric from section 6.2.3.1 tests and utilizing it for this specific test.

Table 6.10: Finding optimal weights – Tests inputs, outputs, and weights.

Model	Input/Output	Weights	Weight Increment	Market
MTDL	[GSPC PRICE, DJI PRICE]	[0.01, 0.99] to [0.99,0.01]	0.01	Bull
MTDL	[GSPC PRICE, NYA PRICE]	[0.01, 0.99] to [0.99,0.01]	0.01	Bear

This series of tests illustrated in Table 6.10 serves as a substitute for the initial research work, which focused on the dynamic adjustment of weights during the training of deep learning models. However, due to the inherent instability and unpredictability of the solution, this approach was set aside and replaced with the current set of experiments.

For comprehensive insights into the original research work concerning the dynamic adjustment of weights, please refer to the details provided in Appendix C.

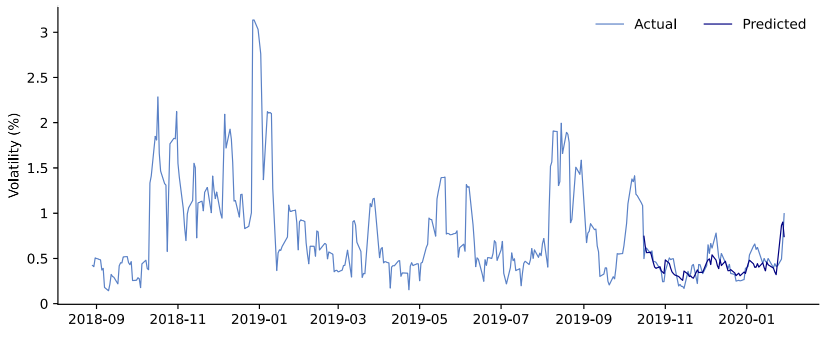
6.3 Results

The results have been structured according to the experimental procedures outlined in the respective section. The sequence of tests conducted is significant as the outcomes from certain tests contribute to the results of other sets. This strategic arrangement enables for effective conclusions and a comprehensive assessment of whether the primary objectives have been achieved. The full list of charts produced by this test is included in Appendix A, Section 2.2.

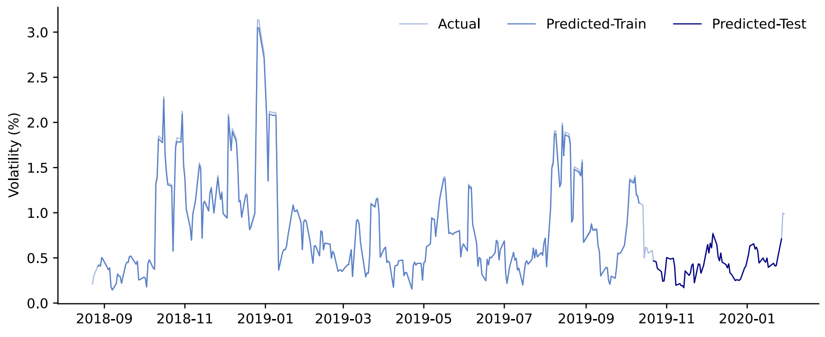
6.3.1 Evaluation through Benchmark Testing

In this series of tests, the primary emphasis will be on predicting the 1-Day future values. This deliberate choice aims to create a highly demanding challenge for deep learning models, thereby facilitating a comprehensive and accurate study.

Upon initial observation, it is evident that the MTDL outperforms both statistical models in all market conditions. While ARIMA competes well in price prediction, it lags behind in returns and volume prediction accuracy. On the other hand, GARCH demonstrates a relatively weaker performance in bear markets which are known to be associated with the most unpredictable market conditions. Figure 6.7 and Figure 6.8 present a sample of prediction charts for GARCH, comparing them to the MTDL model. These charts provide visual insights into the performance differences among the models. ARIMA and MTDL models appear to demonstrate varying effectiveness across particular financial instruments and market conditions. Figure 6.9 and Figure 6.10 vividly demonstrate the superior performance of the DMTL model over the ARIMA model.

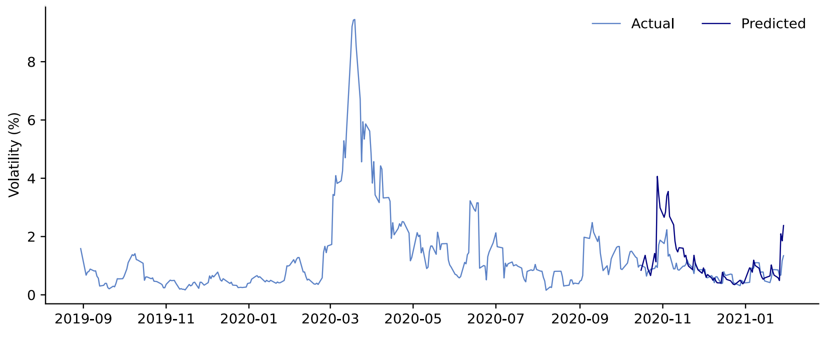


GARCH Model.

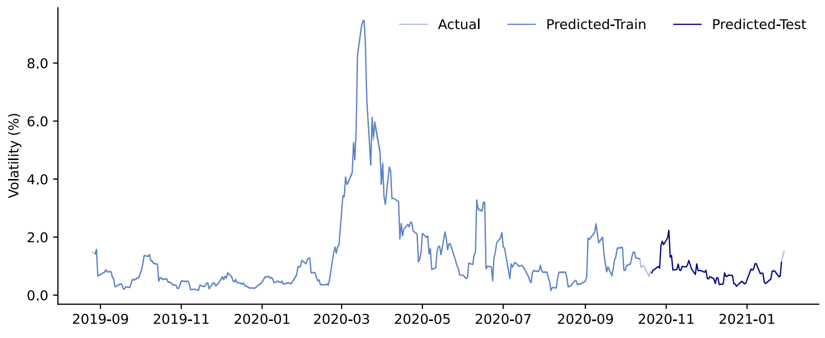


Multi-Task GRU Model.

Figure 6.7: MTDL vs GARCH - Volatility prediction for S&P 500 in bull market.

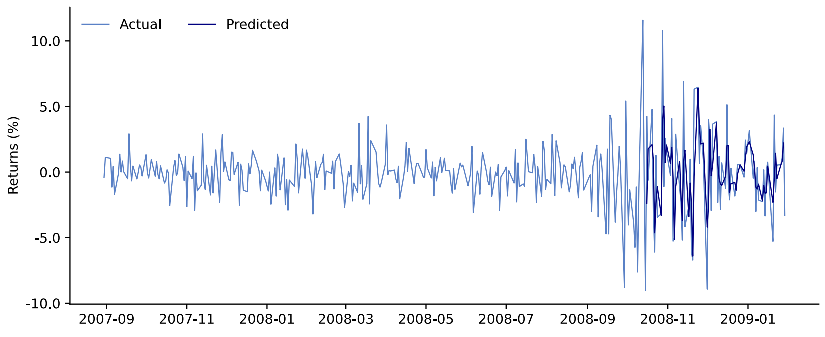


GARCH Model.

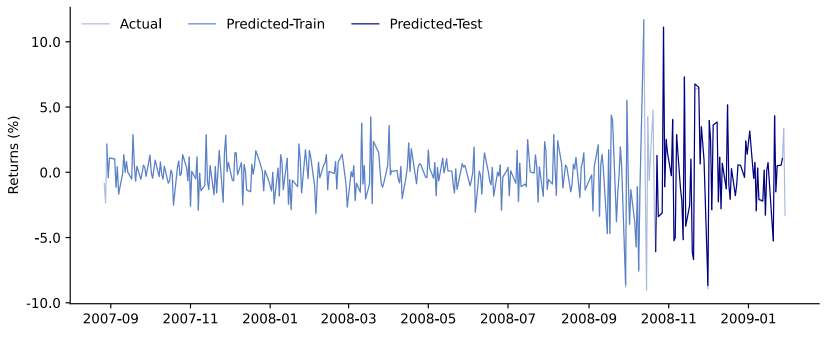


Multi-Task GRU Model.

Figure 6.8: MTDL vs GARCH - Volatility prediction for S&P 500 in volatile market.

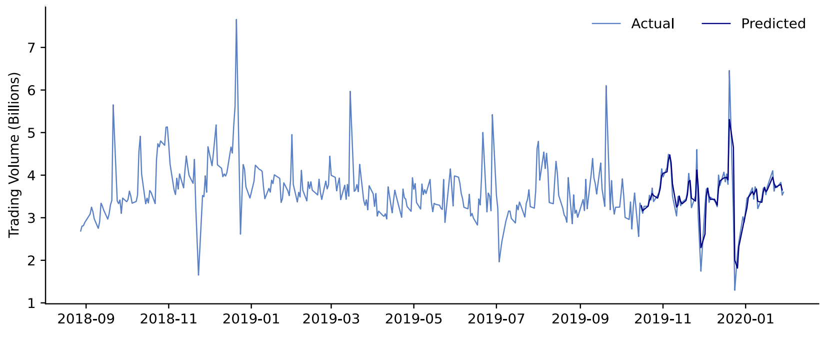


ARIMA Model.

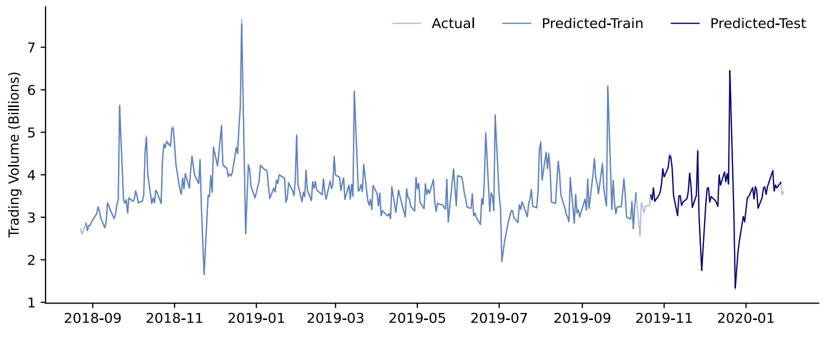


Multi-Task GRU Model.

Figure 6.9: MTDL vs ARIMA - Daily returns prediction for S&P 500 in bear market.

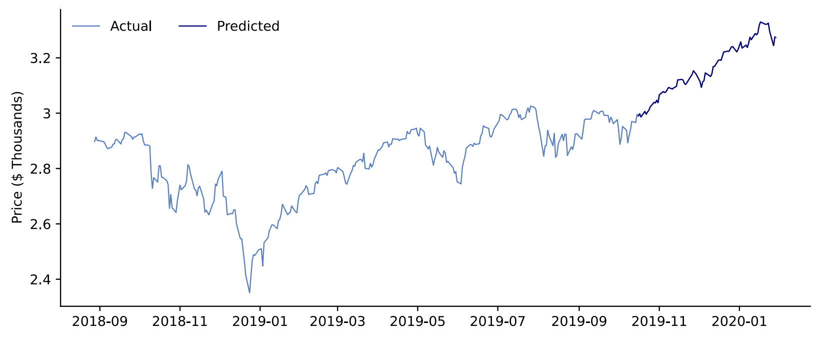


ARIMA Model

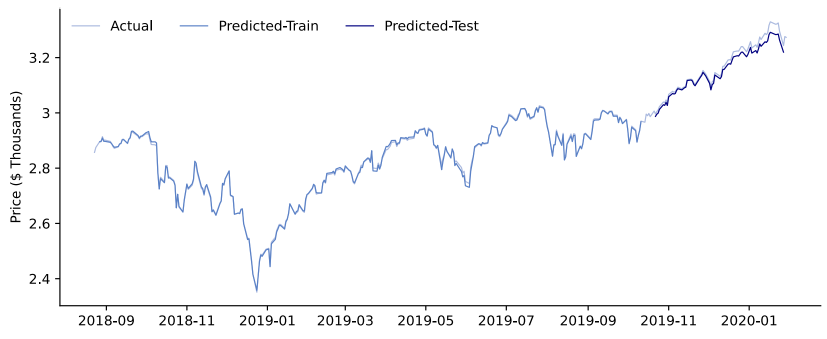


Multi-Task GRU Model

Figure 6.10: MTDL vs ARIMA - Trading volume prediction for S&P 500 in bull market.



ARIMA Model.



Multi-Task LSTM Model.

Figure 6.11: MTDL vs ARIMA - Index price prediction for S&P 500 in bull market.

The Figure 6.11 depicts a comparison between the ARIMA and LSTM MTDL models in predicting the S&P 500 index. Notably, the ARIMA model exhibits a higher level of accuracy and precision in its predictions when contrasted with the deep learning models. This suggests that, for the specific task of forecasting stock price movements, the ARIMA approach outperforms the deep learning approach. Similar results observed for S&P 500 Index price and Down Jones Index prices for LSTM, GRU, and RNN in all bull and bear market conditions, but not during volatile market.

Further analysis is done on the correlation coefficient between index prices for S&P 500 and Dow Jones and it is observed to be positively high (0.99) in bull and bear markets and slightly lower in volatile market (0.92). A full list of correlations is available at Appendix E.

Figure 6.12 and Figure 6.13 showcases the performance across different market conditions for all models. It is evident that all models face challenges in achieving accuracy during bear markets. This difficulty stems from the scarcity of data, making predictions about market direction particularly complex. Despite this, it is noteworthy that ARIMA, GRU, and LSTM exhibit remarkably similar behaviours across all market conditions. Notably, Multi-Task Deep Learning (MTDL) models demonstrate superior performance in volatile markets.

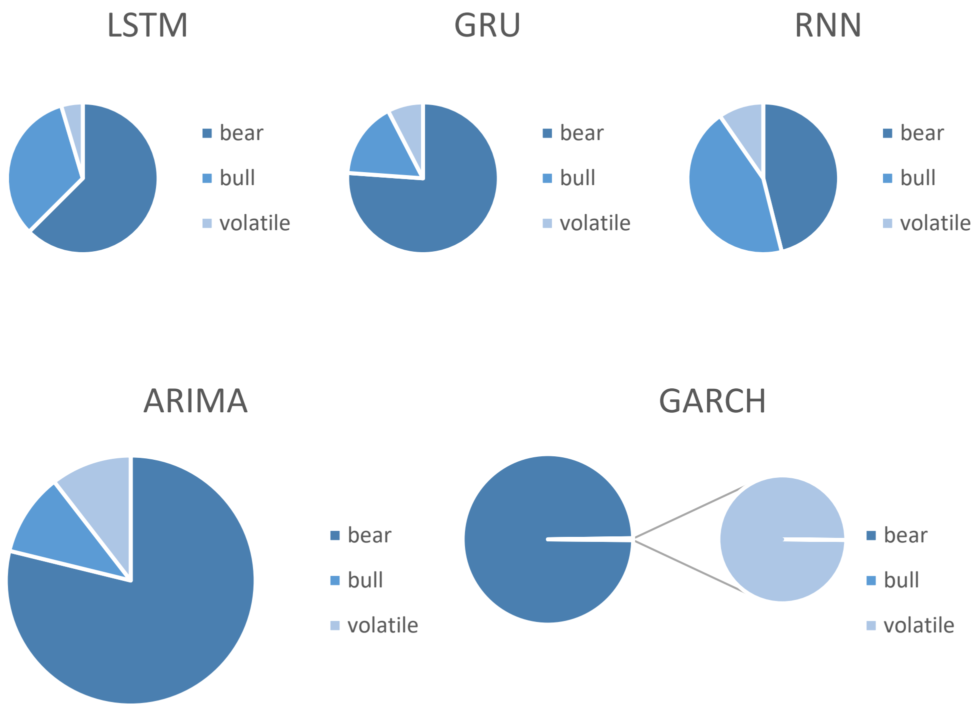


Figure 6.12: Statistical and DL models performance by market conditions.

Upon conducting a comprehensive examination of the outcomes through the lens of Multi-Task Deep Learning (MTDL), a salient and intriguing pattern discernibly emerges, providing valuable insights into the comparative performance of the models under examination. Notably, it becomes conspicuously evident that Recurrent Neural Networks (RNN) lag significantly behind their counterparts, namely Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) models, in terms of predictive capabilities.

This insightful observation harmoniously aligns with a recurring theme that has pervaded the landscape of Single-Task Deep Learning (STDL) models. However, what sets this study apart is its pioneering extension of this corroborated finding into the uncharted territory of Multi-Task Deep Learning (MTDL), thereby illuminating a critical facet of RNN performance. This illumination unequivocally affirms that RNN suboptimal performance is not a characteristic exclusive to STDL models; it permeates into the MTDL domain as well, leaving an indelible mark on its overall efficacy.

Delving into the underlying causes of this discernible performance gap, the root of the issue is traced back to RNN inherent limitation in effectively managing sequence memory. This limitation inherently handicaps RNNs, rendering them less adept in grasping the intricate nuances of sequential data. In stark contrast, both LSTM and GRU models strategically harness this deficiency, adeptly capitalizing on their superior capacity to capture, retain, and manipulate sequence memory. It is this strategic advantage that propels LSTM and GRU models to outperform RNNs across the spectrum of tasks and scenarios evaluated in this study, underscoring the pivotal role played by memory capabilities in the landscape of deep learning models.

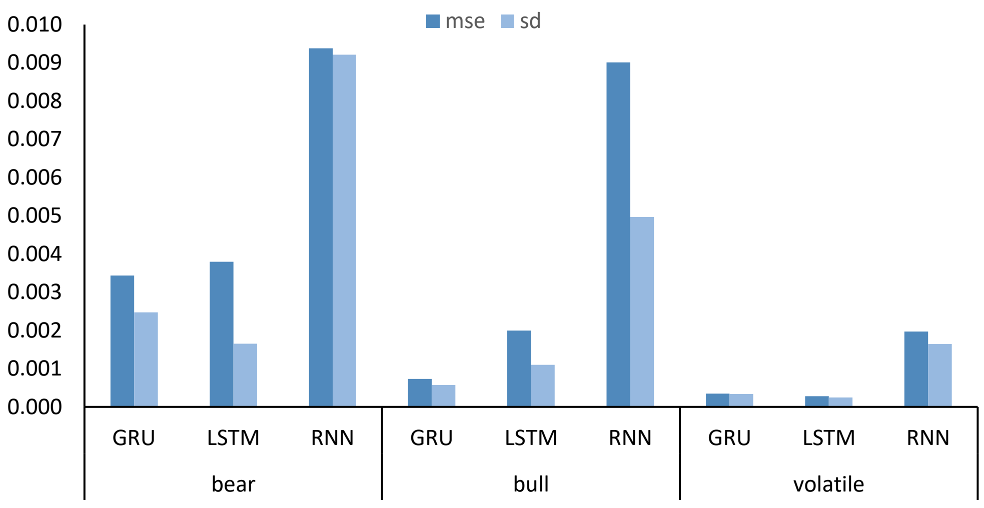


Figure 6.13: Models Performance by market conditions.

In summary, among the models deployed, both LSTM and GRU demonstrate superior performance, with GRU particularly most accurate with exceptional results.

As illustrated in Table 6.7, all tests performed in this section are setup using two inputs with equal weights of [50,50]. In Section 6.3.3, the research undertakes a comprehensive examination by isolating one underperforming task and conducting prediction experiments with different task weight combinations.

Subsequently, in Section 6.3.4, the research investigation is taken a step further by selecting another task that exhibits inferior performance. The objective is to embark on a comprehensive study, experimenting with multiple tasks and varying their weights to identify the optimal combinations, ultimately aiming to enhance prediction performance.

Finally, the results for all tests performed in this set are included in Table 6.11.

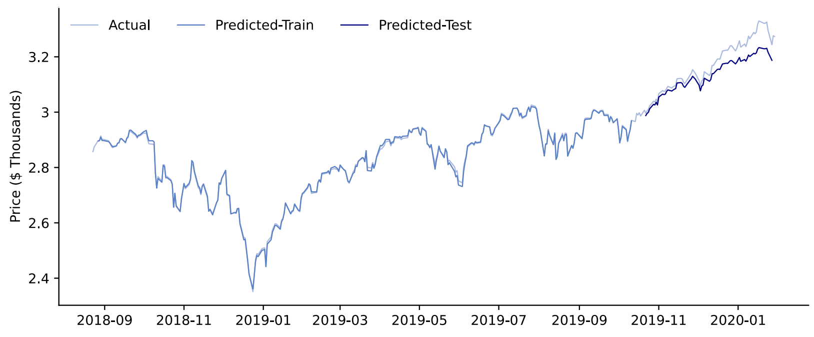
Table 6.11. 1 Day Prediction Accuracy - Benchmark Testing ($\mathbf{10^{-3}}$*).*

Bear Market					
	ARIMA	GARCH	GRU	LSTM	RNN
Dow Jones Price	1.546	-	2.857	0.501	23.389
Dow Jones Returns	2251.697	-	0.851	0.504	3.784
Dow Jones Volatility	21.223	65431.364	2.533	1.110	6.587
Dow Jones Volume	158.403	-	0.396	0.157	1.457
S&P 500 Price	1.944	-	13.469	23.213	24.862
S&P 500 Returns	2313.947	-	1.200	1.082	4.933
S&P 500 Volatility	20.585	84310.574	5.595	3.619	8.958
S&P 500 Volume	86.839	-	0.590	0.188	1.055
Bull Market					
	ARIMA	GARCH	GRU	LSTM	RNN
Dow Jones Price	0.001	-	1.516	10.375	32.507
Dow Jones Returns	145.750	-	0.181	0.058	0.170
Dow Jones Volatility	0.004	57.187	0.109	0.013	0.170
Dow Jones Volume	146.261	-	0.392	0.351	1.118
S&P 500 Price	0.005	-	3.044	4.691	36.469
S&P 500 Returns	130.809	-	0.147	0.029	0.291
S&P 500 Volatility	0.028	38.409	0.162	0.024	0.482
S&P 500 Volume	235.451	-	0.328	0.425	0.881
Volatile Market					
	ARIMA	GARCH	GRU	LSTM	RNN
Dow Jones Price	1.391	-	0.979	0.950	8.000
Dow Jones Returns	202.776	-	0.153	0.077	0.222
Dow Jones Volatility	0.003	241.160	0.056	0.041	0.128
Dow Jones Volume	75.712	-	0.171	0.078	0.405
S&P 500 Price	1.881	-	1.036	0.717	6.191
S&P 500 Returns	256.703	-	0.113	0.054	0.224
S&P 500 Volatility	0.053	207.824	0.088	0.070	0.132
S&P 500 Volume	107.954	-	0.150	0.251	0.445

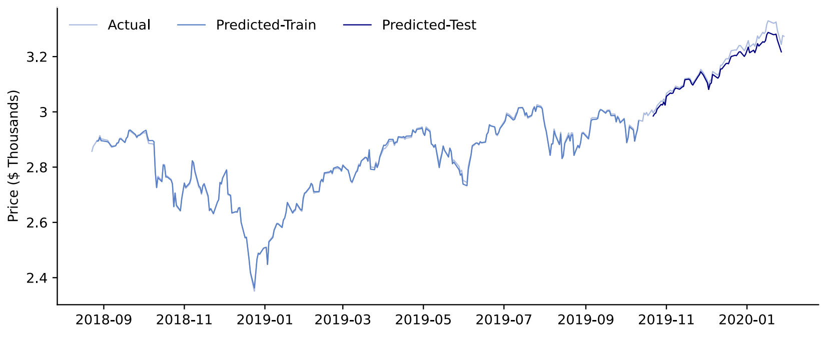
6.3.2 Comparing Multitasking and Single-Task Approaches

Upon observing the results for all future predictions and market conditions, it is evident that Multi-Task Deep Learning (MDTL) outperforms Single-Task Deep Learning (STDL) significantly when the prediction horizon is just one future day. However, as the prediction horizon extends to longer periods, the performance gap between MTDL and STDL narrows due to increased future uncertainty. Nevertheless, MTDL maintains its overall superiority across all market conditions and future prediction horizons. Figure 6.14 depicts performance between STDL and MTDL.

Additionally, it is worth noting that the performance of GRU and LSTM is comparable. However, GRU appears to be superior during bear market conditions, indicating its advantage in handling more challenging market scenarios.



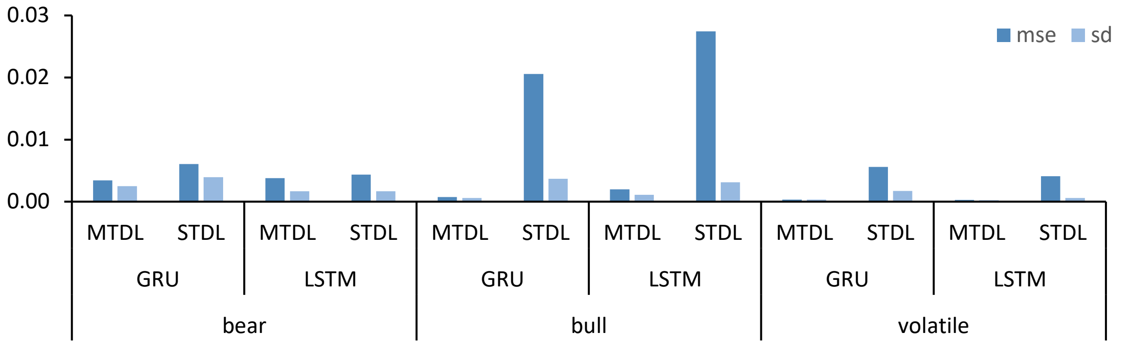
STDL – GRU.



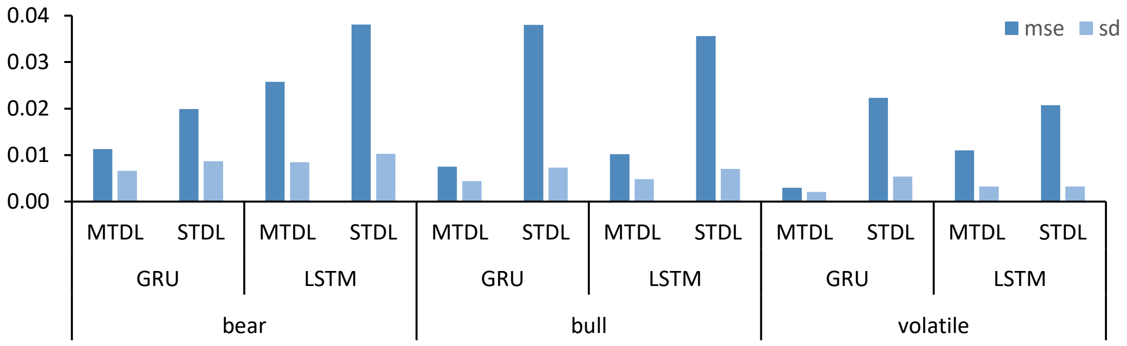
MTDL – GRU.

Figure 6.14: MTDL vs STDL - Index price prediction for S&P 500 in bull market.

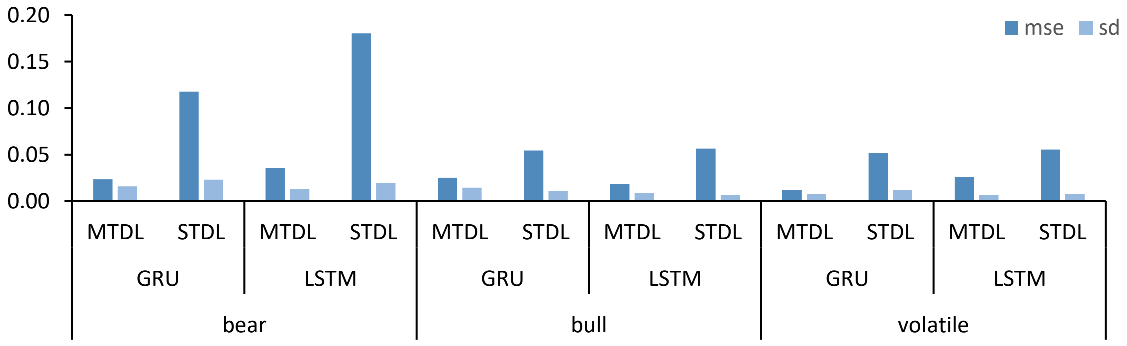
The prediction accuracy charts for MTL and STL in various market conditions and future prediction time horizons are depicted in Figure 6.15. The complete results of all the tests conducted in this set have been organized into separate tables, each corresponding to a specific future prediction horizon. These tables can be found in Table 6.12, Table, 6.13, Table 6.14, and Table 6.15.



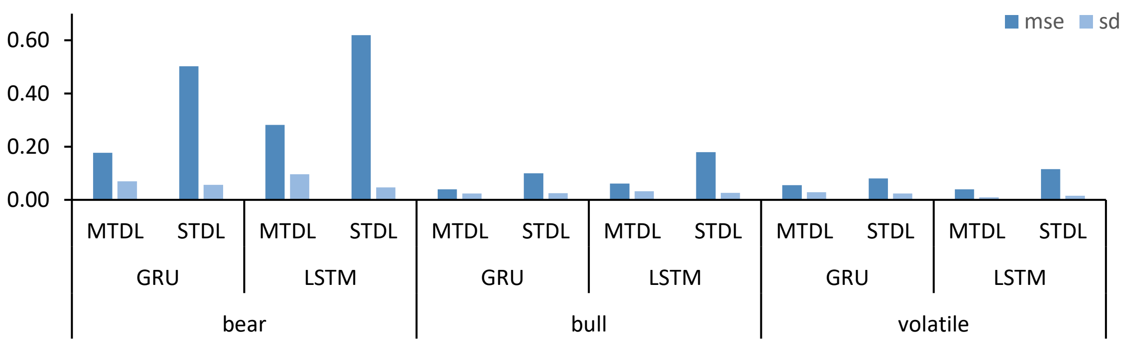
1 Day.



5 Days.



10 Days.



20 Days.

Figure 6.15: Prediction accuracy comparison - MTDL vs STDL.

Table 6.12. MTL vs STL 1-day prediction accuracy ($\mathbf{10^{-4}}$).

Bear Market

	GRU				LSTM			
	MTDL		STDL		MTDL		STDL	
	MSE	SD	MSE	SD	MSE	SD	MSE	SD
Dow Jones Price	28.57	35.79	73.13	61.74	5.01	6.96	24.00	6.59
Dow Jones Returns	8.51	5.57	40.08	22.19	5.04	3.33	23.60	7.43
Dow Jones Volatility	25.33	17.14	54.93	39.66	11.10	5.62	52.13	27.54
Dow Jones Volume	3.96	5.97	9.94	8.95	1.57	0.99	7.96	4.43
S&P 500 Price	134.69	91.38	199.37	118.26	232.13	83.30	165.22	64.32
S&P 500 Returns	12.00	4.37	27.76	9.54	10.82	7.56	39.78	10.53
S&P 500 Volatility	55.95	26.80	67.52	47.63	36.19	23.08	32.26	10.64
S&P 500 Volume	5.90	11.06	10.49	6.87	1.88	1.19	4.85	4.17

Bull Market

	GRU				LSTM			
	MTDL		STDL		MTDL		STDL	
	MSE	SD	MSE	SD	MSE	SD	MSE	SD
Dow Jones Price	15.16	7.91	586.52	101.75	103.75	60.04	568.81	90.87
Dow Jones Returns	1.81	1.51	2.65	1.72	0.58	0.21	0.53	0.32
Dow Jones Volatility	1.09	1.01	1.79	2.28	0.13	0.09	0.59	0.17
Dow Jones Volume	3.92	1.23	17.35	7.28	3.51	1.27	21.42	4.15
S&P 500 Price	30.44	29.80	1019.18	175.09	46.91	24.83	1586.93	152.42
S&P 500 Returns	1.47	1.33	2.03	1.48	0.29	0.12	1.10	0.89
S&P 500 Volatility	1.62	1.29	2.02	3.84	0.24	0.20	0.54	0.14
S&P 500 Volume	3.28	1.66	13.01	3.21	4.25	1.06	15.50	2.11

Volatile Market

	GRU				LSTM			
	MTDL		STDL		MTDL		STDL	
	MSE	SD	MSE	SD	MSE	SD	MSE	SD
Dow Jones Price	9.79	9.70	123.26	41.42	9.50	13.06	53.27	12.54
Dow Jones Returns	1.53	1.06	2.23	1.84	0.77	0.44	1.36	0.62
Dow Jones Volatility	0.56	0.42	1.05	0.97	0.41	0.23	0.39	0.13
Dow Jones Volume	1.71	1.67	4.90	2.35	0.78	0.39	1.37	1.07
S&P 500 Price	10.36	11.09	297.33	86.08	7.17	3.76	261.81	29.94
S&P 500 Returns	1.13	0.92	2.74	1.87	0.54	0.26	0.95	0.50
S&P 500 Volatility	0.88	0.81	1.02	1.31	0.70	0.28	0.79	0.29
S&P 500 Volume	1.50	1.29	16.72	3.32	2.51	1.53	9.90	2.73

Table 6.13. MTL vs STL 5-day prediction accuracy ($\mathbf{10^{-4}}$).

Bear Market								
	GRU				LSTM			
	MTDL		STDL		MTDL		STDL	
	MSE	SD	MSE	SD	MSE	SD	MSE	SD
Dow Jones Price	83.33	70.93	206.89	126.05	275.69	169.09	418.93	124.08
Dow Jones Returns	23.19	3.76	66.65	21.50	87.79	9.17	207.54	21.56
Dow Jones Volatility	100.00	76.65	280.32	122.86	256.37	41.41	925.77	306.14
Dow Jones Volume	25.59	11.21	61.80	8.64	124.89	31.07	60.11	5.66
S&P 500 Price	244.67	209.69	541.06	203.20	641.90	333.52	553.54	165.18
S&P 500 Returns	92.87	28.41	83.01	20.47	216.46	13.38	200.01	22.37
S&P 500 Volatility	291.66	95.19	255.22	148.62	399.61	67.22	579.59	152.71
S&P 500 Volume	44.27	36.22	99.49	43.07	59.72	13.36	101.62	21.70
Bull Market								
	GRU				LSTM			
	MTDL		STDL		MTDL		STDL	
	MSE	SD	MSE	SD	MSE	SD	MSE	SD
Dow Jones Price	260.46	152.71	607.46	132.82	285.35	105.86	322.06	75.43
Dow Jones Returns	3.55	1.19	3.05	0.96	5.15	5.21	6.37	1.33
Dow Jones Volatility	3.75	2.74	18.73	6.09	21.01	3.43	15.51	2.25
Dow Jones Volume	55.95	18.30	115.19	39.07	68.54	21.87	206.57	38.06
S&P 500 Price	216.93	162.57	2163.70	371.24	374.73	218.08	2064.41	364.41
S&P 500 Returns	3.35	1.32	3.43	1.03	5.65	1.83	6.78	1.28
S&P 500 Volatility	5.31	3.21	15.95	5.49	8.98	3.48	11.05	3.80
S&P 500 Volume	50.57	13.10	115.14	27.32	48.81	24.72	214.73	75.34
Volatile Market								
	GRU				LSTM			
	MTDL		STDL		MTDL		STDL	
	MSE	SD	MSE	SD	MSE	SD	MSE	SD
Dow Jones Price	78.53	49.22	536.57	112.82	91.80	33.43	392.51	66.92
Dow Jones Returns	6.81	2.50	13.58	3.93	120.29	17.26	61.36	6.98
Dow Jones Volatility	8.48	5.32	24.23	8.22	36.54	1.89	38.30	8.99
Dow Jones Volume	16.15	5.88	24.96	10.19	29.48	10.01	53.59	5.72
S&P 500 Price	105.82	93.63	1152.38	283.57	489.78	178.02	1009.75	147.80
S&P 500 Returns	6.78	2.70	9.49	3.22	65.39	10.51	39.82	7.91
S&P 500 Volatility	8.59	1.32	11.76	4.37	38.21	4.49	13.74	3.30
S&P 500 Volume	6.13	2.89	11.67	4.73	12.06	5.49	48.36	14.40

Table 6.14. MTL vs STL 10-day prediction accuracy ($\mathbf{10^{-3}}$).

Bear Market								
	GRU				LSTM			
	MTDL		STDL		MTDL		STDL	
	MSE	SD	MSE	SD	MSE	SD	MSE	SD
Dow Jones Price	22.82	23.28	58.14	16.62	43.86	28.84	171.46	30.57
Dow Jones Returns	1.90	0.89	225.02	21.79	11.46	1.12	227.92	11.92
Dow Jones Volatility	60.81	49.00	245.82	63.98	111.63	20.86	530.67	50.67
Dow Jones Volume	4.64	2.48	16.85	3.75	23.64	8.27	16.39	1.62
S&P 500 Price	58.79	28.17	140.62	32.59	58.97	32.19	171.80	29.86
S&P 500 Returns	7.55	3.10	125.47	21.73	16.48	1.60	125.83	12.41
S&P 500 Volatility	26.64	17.61	118.66	20.94	8.71	4.81	183.97	14.09
S&P 500 Volume	3.56	1.31	10.50	4.32	8.30	2.94	16.06	2.90

Bull Market								
	GRU				LSTM			
	MTDL		STDL		MTDL		STDL	
	MSE	SD	MSE	SD	MSE	SD	MSE	SD
Dow Jones Price	121.13	73.08	75.05	28.37	69.62	22.90	32.65	8.96
Dow Jones Returns	0.63	0.24	0.28	0.11	6.26	1.77	0.98	0.23
Dow Jones Volatility	0.42	0.17	2.37	0.57	5.68	0.64	4.12	0.46
Dow Jones Volume	7.57	3.23	25.89	11.13	27.52	19.08	72.36	4.93
S&P 500 Price	63.76	36.54	298.12	34.21	8.04	4.60	265.30	21.15
S&P 500 Returns	0.49	0.19	0.49	0.17	2.32	0.47	0.39	0.12
S&P 500 Volatility	1.07	0.59	1.43	0.50	2.35	0.50	2.69	0.35
S&P 500 Volume	5.38	2.40	31.36	10.35	26.54	21.52	74.17	16.99

Volatile Market								
	GRU				LSTM			
	MTDL		STDL		MTDL		STDL	
	MSE	SD	MSE	SD	MSE	SD	MSE	SD
Dow Jones Price	30.06	25.86	137.52	35.01	25.74	8.62	135.60	22.08
Dow Jones Returns	1.69	1.30	6.49	1.26	17.69	1.53	16.75	2.57
Dow Jones Volatility	0.66	0.40	4.11	1.13	3.59	0.73	6.84	0.78
Dow Jones Volume	2.01	0.69	3.37	1.10	12.18	2.48	4.80	1.18
S&P 500 Price	55.83	29.99	258.01	56.18	132.83	30.88	258.83	29.30
S&P 500 Returns	1.03	0.81	1.70	0.71	5.96	1.22	5.20	0.98
S&P 500 Volatility	2.44	0.52	2.87	1.32	6.56	1.26	4.66	0.86
S&P 500 Volume	1.20	0.66	2.86	0.86	4.50	4.27	11.29	2.21

Table 6.15. MTL vs STL 20-day prediction accuracy (10^{-3}).

Bear Market

	GRU				LSTM			
	MTDL		STDL		MTDL		STDL	
	MSE	SD	MSE	SD	MSE	SD	MSE	SD
Dow Jones Price	564.35	193.93	1452.16	104.60	994.49	279.38	2126.33	153.95
Dow Jones Returns	52.64	12.23	116.54	22.27	35.50	6.41	142.92	12.93
Dow Jones Volatility	159.84	61.07	271.50	49.31	242.14	33.66	300.24	28.12
Dow Jones Volume	15.39	6.24	24.58	6.12	41.29	5.55	33.92	3.97
S&P 500 Price	501.69	219.70	1888.00	206.74	882.59	427.18	2121.85	147.74
S&P 500 Returns	22.21	9.97	58.69	16.02	14.87	3.27	76.65	8.23
S&P 500 Volatility	80.06	50.53	175.24	40.59	21.59	13.89	114.51	17.29
S&P 500 Volume	18.05	7.05	28.87	5.96	23.35	4.88	39.40	6.30

Bull Market

	GRU				LSTM			
	MTDL		STDL		MTDL		STDL	
	MSE	SD	MSE	SD	MSE	SD	MSE	SD
Dow Jones Price	162.39	99.99	157.65	35.87	103.21	93.28	190.84	94.75
Dow Jones Returns	0.84	0.29	0.19	0.13	8.01	1.11	2.18	0.39
Dow Jones Volatility	3.97	2.70	12.81	5.33	20.25	3.82	12.67	1.59
Dow Jones Volume	15.70	10.91	62.93	21.88	106.06	21.71	153.80	9.59
S&P 500 Price	117.32	71.04	480.72	121.20	99.93	59.85	817.97	70.44
S&P 500 Returns	0.46	0.19	0.65	0.20	4.13	0.62	0.99	0.35
S&P 500 Volatility	2.66	2.58	6.55	3.19	9.00	3.11	11.41	1.54
S&P 500 Volume	14.30	4.22	75.82	15.80	137.38	71.02	247.60	33.69

Volatile Market

	GRU				LSTM			
	MTDL		STDL		MTDL		STDL	
	MSE	SD	MSE	SD	MSE	SD	MSE	SD
Dow Jones Price	126.24	67.16	179.27	75.83	36.30	13.53	305.08	43.13
Dow Jones Returns	1.66	1.29	13.95	5.04	10.74	1.65	22.24	2.73
Dow Jones Volatility	2.43	0.74	5.80	2.33	3.68	1.33	13.43	1.50
Dow Jones Volume	2.28	0.78	6.32	0.84	14.06	2.78	10.58	1.30
S&P 500 Price	303.68	155.74	425.51	102.01	222.26	44.37	531.60	66.66
S&P 500 Returns	3.10	2.03	2.07	1.14	7.39	1.11	5.24	1.03
S&P 500 Volatility	2.48	1.03	1.78	0.50	4.93	1.55	3.87	0.36
S&P 500 Volume	3.78	2.07	9.26	2.17	16.46	8.60	28.59	4.68

6.3.3 The Art of Multitasking – Finding the Optimal Task Combination

The results obtained support the hypothesis that incorporating additional outputs in Multi-Task Deep Learning (MTDL) can enhance prediction accuracy. This improvement is achieved by iteratively adjusting the model parameters based on the error gradient, which leads to a reduction in loss and improved predictions. It was also noted that various combinations of metrics or features can yield similar effects on the error gradient, thus resulting in improved prediction accuracy. The results obtained from this set of tests are depicted in Figure 6.16.

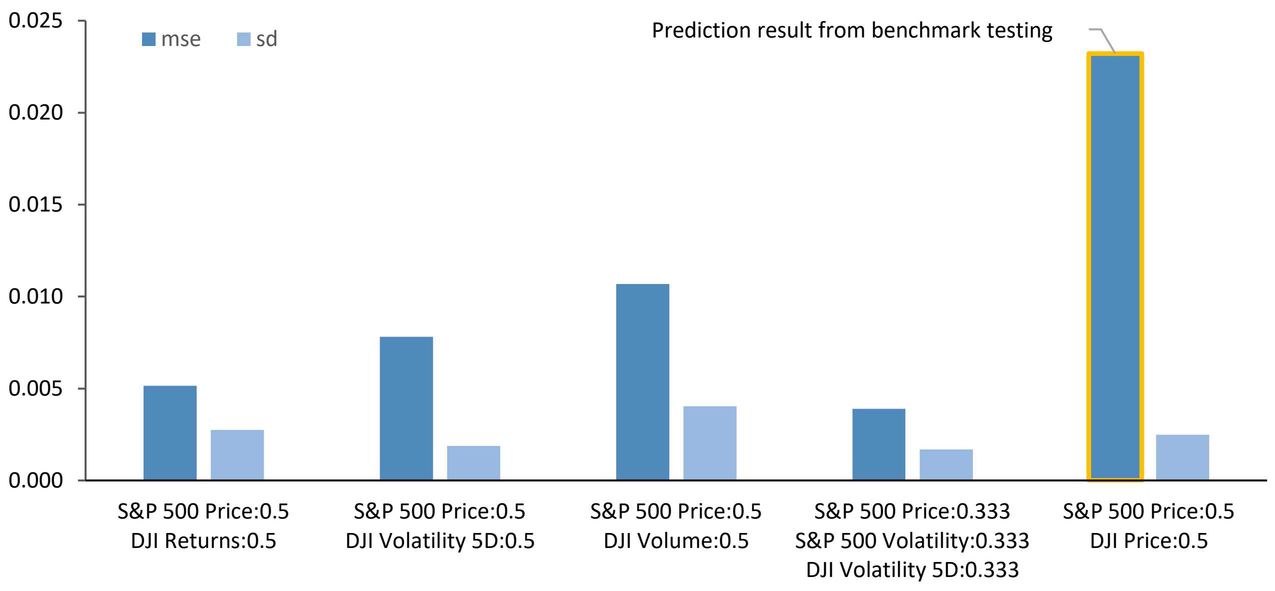


Figure 6.16: Finding the Optimal Task Combinations.

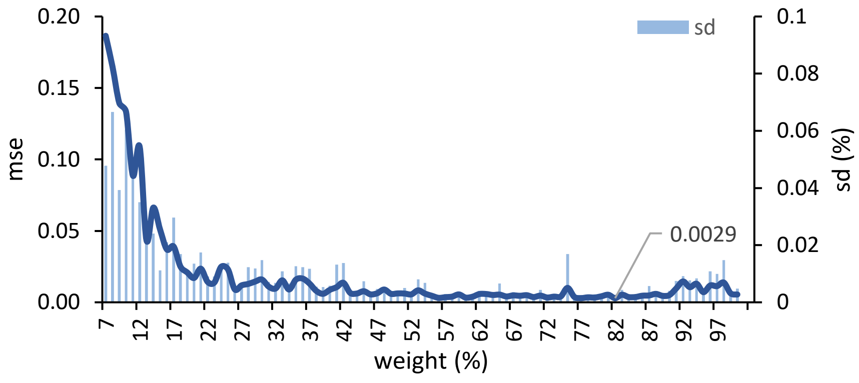
6.3.4 Fine-Tuning the Loss Function: Determining Optimal Weights

In the benchmark testing phase, the Mean Squared Error (MSE) for S&P 500 and Dow Jones prices within a bull market stood at 0.00469 and 0.01037, respectively, using the weight setup [GSPC_PRICE, DJI_PRICE: 0.5, 0.5]. However, following the optimization of weights, the MSE exhibited significant improvements. For S&P 500, with weights [0.82, 0.18], the MSE decreased by approximately 37% to 0.00293. Similarly, for Dow Jones, using weights [22, 78], the MSE experienced a substantial reduction of 78.6% to 0.00390.

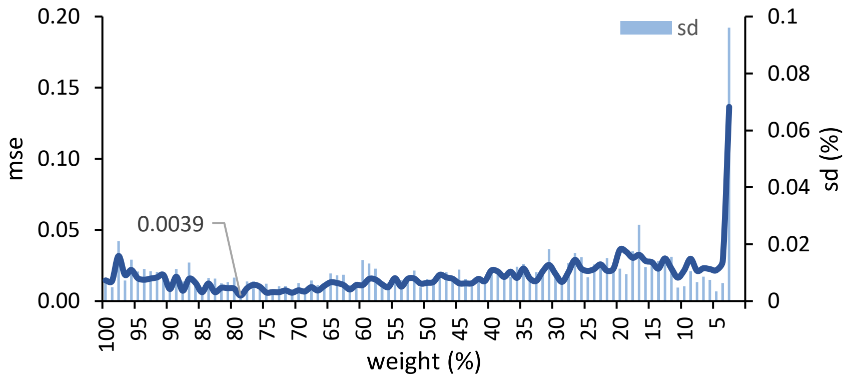
To validate these findings, an additional test was conducted using the weight setup [GSPC_PRICE and NYA_PRICE: 0.5, 0.5] within a bear market scenario. This test yielded similar outcomes, with a reduction of 13% observed in the case of S&P 500 (MSE decreasing from 0.003178 to 0.00275), and a substantial 59% reduction for the New York Stock Exchange (MSE decreasing from 0.00281 to 0.00115).

For a thorough and insightful examination of the tests outcomes, the focus is directed to Figure 6.17, which pertains to the bull market test, and Figure 6.18, which focuses on the bear market test. These figures encapsulate the nuanced details and patterns uncovered during the extensive evaluation process.

This noteworthy discovery lends credence to the hypothesis that metric weights need not be uniform or weighted equally. Instead, it underscores the paramount importance of discerning the optimal weight relative to the specific test metrics deployed to nourish the model. In essence, this emphasizes the tailored and context-sensitive nature of metric weighting in the realm of Multi-Task Deep Learning (MTDL), where precision in weight assignment can significantly impact predictive performance across diverse market conditions.

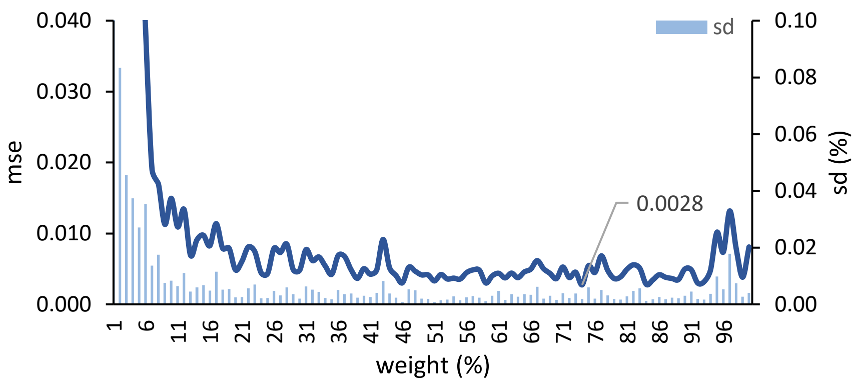


S&P 500 index price optimal weight.

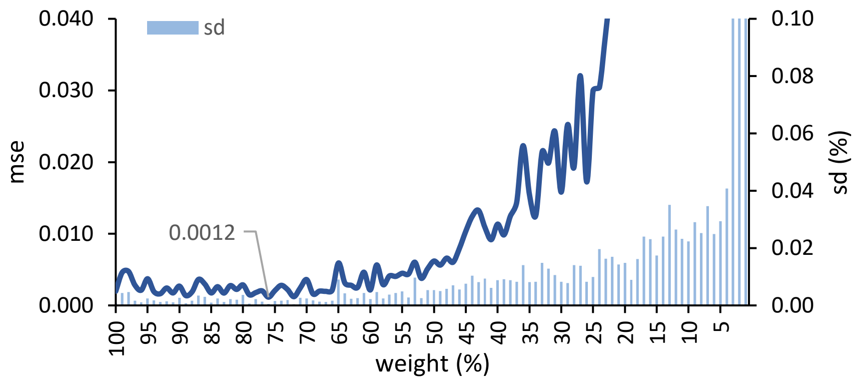


Dow Jones index price optimal weight.

Figure 6.17: Loss function tuning – Finding optimal weights, bull market.



S&P 500 index price optimal weight.



NYSE index price optimal weight.

Figure 6.18: Loss function tuning – Finding optimal weights, bear market.

6.4 Conclusions

The research introduces a novel implementation of Multi-Task Deep Learning (MTDL), aiming to significantly enhance the prediction accuracy of key stock market metrics. The study commences with three primary hypotheses: Firstly, that multi-task learning can improve the prediction accuracy of deep learning models. Secondly, that varying combinations of metrics can lead to distinct levels of accuracy, thereby determining the right metrics for specific market conditions to achieve optimal performance. Lastly, the thesis examines the impact of adjusting the weight of metrics fed into the MTDL model.

To ensure the maximum accuracy of the proposed model, extensive testing was conducted across different market conditions, using various prediction horizons, and deploying two commonly used deep learning RNN architectures, LSTM and GRU. Each test was repeated ten times for further validation and confirmation.

The results observed have confirmed that multi-task deep leaning prediction accuracy is improved by using a shared layer of the metrics inputs. It is also observed that adding more input/output to the model can significantly improve the prediction performance.

Although ARIMA can exhibit superiority in price prediction during bull markets, it falls behind MTDL in predicting returns and volume across all market conditions. This disparity can be attributed to the trend in the average of data points, which is a fundamental concept of the ARIMA model.

On the other hand, GARCH did not perform well in the experiments, primarily due to the deliberately selected small volume of data used for short time horizons. This was done to stress-test the performance of all models when dealing with limited data, and it impacted the accuracy of the GARCH model.

As validated in references [47], [50], [51], deep learning models exhibit superiority in time series data prediction owing to their memory capabilities and effective handling of previously processed sequence data. In this research, it is established from results obtained that GRU outperforms LSTM, primarily due to the horizon of data used in the experiments [78], [79].

Chapter 7

Discussion and Conclusions

This comprehensive research has examined three key dimensions, each contributing significantly to the advancement of stock market prediction and the potential for enhanced financial modelling. These dimensions have encompassed the utilisation of Deep Learning, the introduction of Multivariate Deep Learning (MDL), and the innovative concept of Multi-Task Deep Learning (MTDL). Within this multifaceted research, several noteworthy innovations have taken centre stage as follows:

Comprehensive Model Testing: This research set out to conduct a rigorous battery of tests spanning diverse stock market conditions, including bear, bull, and volatile scenarios. This comprehensive approach ensured that all models underwent scrutiny across the entire spectrum of market dynamics, providing a holistic view of their performance.

Varied Time Horizons: To assess model performance with precision, predictions were made across different time horizons, encompassing 1-day, 5-day, 10-day, and 20-day forecasts. This multifaceted evaluation captured the capability of models to adapt to varying prediction horizons, a crucial aspect of their utility.

Multivariate Input Combinations: The research has examined the impact of utilizing multiple combinations of inputs during multivariate testing. This extensive examination has evaluated how different input combinations influenced the predictive capabilities of the model, shedding light on the most effective data configurations.

Weight Variations in Multi-Task Learning: In the realm of Multi-Task Deep Learning (MTDL), this research has introduced the concept of using different weights during multi-task testing. By examining the effects of varying weights on loss functions and overall MTDL model performance, this innovation uncovers valuable insights into optimizing the model learning process.

Task Diversity in Multitasking Models: Within the MTDL framework, various combinations of tasks were examined during multitasking model testing. This approach sought to understand how the inclusion of various tasks impacts the overall performance of the multi-task model, providing valuable guidance on task selection and model optimization.

These innovative elements collectively contributed to a comprehensive and insightful analysis of stock market prediction and financial modelling. Through systematic testing under diverse time horizons, input permutations, weight adjustments, and task diversity, this work has paved the way for enhanced predictive models and a deeper understanding of their adaptability across multifaceted market conditions.

7.1 Utilizing Deep Learning Networks for Market Prediction

The first strand of this research ventures into the realm of deep learning, in particular, LSTM, seeking to challenge the traditional stronghold of classical statistical models such as GARCH. This research is multifaceted, aiming to assess the LSTM potential in predicting market volatility, optimising trading portfolios, and refining options pricing models.

The benchmark testing provided a resounding confirmation of the initial hypothesis – that LSTM, powered by deep learning, improves over the performance of GARCH in volatility forecasting. These finding underscores the adaptability of LSTM for intra-day volatility modelling, particularly due to its capability in detecting short-term market influencers.

7.2 Improving Performance by Utilizing the Multivariate Deep Learning (MDL) Architecture

The second strand of this research extended to the realm of multivariate LSTM architectures, showcasing their efficacy in enhancing volatility forecasting. As evidenced across both bear and bull markets, the inclusion of additional asset prices within the multivariate input has led to marked performance improvements, especially in 1-day and 5-day predictions. The introduction of a lag window proved to be a critical factor, particularly for enhancing LSTM prediction accuracy amidst the turbulence of bear markets. It is worth noting that the most accurate predictions often arise from input combinations characterised by strong correlations, whether they are positive or negative.

The next steps included further enhancing the predictive accuracy of LSTM and extending its forecasting horizon. This entailed optimising the multivariate input by examining various lagging periods and integrating new types of time series data, including news and weather information. Furthermore, delving into the correlation between test stocks and the correlation among feature inputs unveiled additional opportunities for fine-tuning the multivariate models.

7.3 Introducing the Multi-Task Deep Learning (MTDL) for Enhanced Predictions

The third strand of the research introduced the novel concept of Multi-Task Deep Learning (MTDL), aimed at substantially enhancing the accuracy of stock market predictions. With a foundation based on three primary hypotheses, the study examined the various benefits of MTDL.

The rigorous testing, encompassing diverse market conditions, varying prediction horizons, and the utilisation of two prevalent deep learning RNN architectures, LSTM and GRU, resulted in compelling insights. Multi-Task deep learning with the shared layer of metric inputs, improves prediction accuracy significantly. Moreover, the results have shown that expanding the input/output dimensions of the model can yield substantial performance enhancements.

In the context of conventional models, ARIMA has stood out for price prediction during bull markets yet lagged behind MTDL in predicting returns and volume across diverse market conditions. The inherent trend-capturing capability of ARIMA constrains its adaptability to intricate market dynamics.

On a different note, GARCH performance waned, primarily due to deliberately restricting the data volume for short time horizons. This constraint, intentionally introduced for robust stress-testing, notably impacted the model accuracy.

Consistent with previous research findings, deep learning models, given their memory capabilities and adeptness in handling sequential data, tended to show superior performance. This research revealed GRU outperformance over LSTM, primarily attributed to the extended data horizon utilised.

MTDL emerged as a frontrunner in most market conditions, with the potential to most significant performance gains in stock market predictions. However, additional research is warranted to devise optimal strategies for metric selection. Notably, the presence of highly correlated inputs did not consistently yield improved performance.

Further complexity arises when determining the optimal weights for MTDL in scenarios involving more than two stocks. Dynamic weight adjustments during the learning process stand as a promising avenue for minimising gradient errors and enhancing model performance.

Multitasking represents a relatively novel area in stock market analysis, leaving ample space for enhancements and research. Preliminary experiments have shown promise in improving the gradient descent behaviour through such weight adjustments. These investigations will contribute to the ongoing evolution of multi-task deep learning in stock market analysis. The sample code of the experiment has been included to Appendix D.

7.4 Future Work

Future work could focus on further refining LSTM prediction accuracy and extending the prediction horizon. This will include fine-tuning the multivariate input by examining various lagging periods and introducing new categories of time series input data, such as news and weather information. Additionally, forthcoming research could leverage the proposed LSTM model to predict implied volatility which is a critical parameter in options pricing models.

MTDL has demonstrated its superior performance in almost all market conditions. However, additional research is needed to determine the most effective approach for selecting the metrics to be fed into the model. It is noted that highly correlated inputs do not always lead to improved performance.

While finding the optimal weights for MTDL was relatively straightforward when using two metrics, the complexity increased when dealing with more than two stocks. Further investigation was required to dynamically adjust the weights during the learning process to help the model achieve minimum gradient error and enhance its performance.

Future research could address the need for a detailed examination of the correlation between tasks and input data, aiming to develop a feature-selection system that could be seamlessly integrated into the feature input list of multi-task models before the training phase.

Additionally, this would have to include optimisation studies aimed at dynamic allocating weights to features during the training process.

References

[1] D. Colander *et al.*, "THE FINANCIAL CRISIS AND THE SYSTEMIC FAILURE OF THE ECONOMICS PROFESSION," *Critical Review*, vol. 21, no. 2–3, pp. 249–267, Jun. 2009, Doi: 10.1080/08913810902934109.

[2] S. Supervisors Group, "Risk Management Lessons from the Global Banking Crisis of 2008 Senior Supervisors Group," 2009.

[3] P. Jorion, *Financial Risk Manager Handbook*. John Wiley & Sons, 2003.

[4] R. Bhowmik and S. Wang, "Stock Market Volatility and Return Analysis: A Systematic Literature Review," *Entropy*, vol. 22, no. 5, May 2020, doi: 10.3390/E22050522.

[5] C. Alexander, *Market Risk Analysis Volume II Practical Financial Econometrics*. 2008.

[6] George E. P. Box, Gwilym M. Jenkins, Gregory C. Reinsel, and Greta M. Ljung, *Time Series Analysis: Forecasting and Control*, 5th Edition. 2015.

[7] E. Chong, C. Han, and F. C. Park, "Deep learning networks for stock market analysis and prediction: Methodology, data representations, and case studies," *Expert Syst Appl*, vol. 83, pp. 187–205, 2017, doi: 10.1016/j.eswa.2017.04.030.

[8] W. Long, Z. Lu, and L. Cui, "Deep learning-based feature engineering for stock price movement prediction," *Knowl Based Syst*, vol. 164, pp. 163–173, Jan. 2019, doi: 10.1016/J.KNOSYS.2018.10.034.

[9] E. Easterling, "Volatility In Perspective," 2007. [Online]. Available: www.CrestmontResearch.com

[10] Z. , Bodie, A. Kane, and A. Marcus, *Investments*, Eighth edition. McGraw-Hill/Irwin, 2008.

[11] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. 2015.

[12] M. Nielsen, "Neural Networks and Deep Learning", Accessed: Aug. 11, 2023. [Online]. Available: http://neuralnetworksanddeeplearning.com

[13] L. Harris, *Trading and Exchanges: Market Microstructure for Practitioners*. Oxford University Press, 2003.

[14] S. Benninga, *Financial Modeling*, Third. The MIT Press, 2008.

[15] F. Black and M. Scholes, "The Pricing of Options and Corporate Liabilities." Accessed: Jan. 22, 2019. [Online]. Available: https://www.cs.princeton.edu/courses/archive/fall09/cos323/papers/black_scholes73.pdf

[16] W. F. Sharpe, "The Arithmetic of Active Management," *https://doi.org/10.2469/faj.v47.n1.7*, vol. 47, no. 1, pp. 7–9, Jan. 2018, doi: 10.2469/FAJ.V47.N1.7.

[17] P. Wilmott, *Paul Wilmott Introduces Quantitative Finance*, Second. John Willey & Sons, Ltd, 2007.

[18] C. Alexander, *Market Risk Analysis, Volume IV, Value at Risk Models*. JohnWiley & Sons Ltd, 2009.

[19] C. Marshall and M. Siegel, "Value at Risk: Implementing a Risk Measurement Standard," 1996, Accessed: Jun. 15, 2017. [Online]. Available: http://ssrn.com/abstract=1212

[20] C. Alexander, *Market Models*. John Willey & Sons Ltd, 2003.

[21] C. Alexander, *Market Risk Analysis - Volume III*. 2008. doi: 10.1007/s13398-014-0173-7.2.

[22] P. Jorion, "VALUE AT RISK: The New Benchmark for Managing Financial Risk".

[23] R. F. Engle, "Autoregressive Conditional Heteroscedasticity with Estimates of the Variance of United Kingdom Inflation," *Econometrica*, vol. 50, no. 4, p. 987, Jul. 1982, doi: 10.2307/1912773.

[24] C. Alexander, *Market Risk Analysis, Practical Financial Econometrics*. 2008.

[25] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation (No. ICS-8506).," *California Univ San Diego La Jolla Inst For Cognitive Science*, vol. 1, pp. 318–362, 1986, doi: 10.1016/B978-1-4832-1446-7.50035-2.

[26] S. Hochreiter, “The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions.” Accessed: Jan. 22, 2019. [Online]. Available: https://www.bioinf.jku.at/publications/older/2304.pdf

[27] S. Hochreiter and J. Urgen Schmidhuber, “Long Short Term Memory,” *Neural Comput*, vol. 9, no. 8, pp. 1735–1780, 1997, doi: 10.1162/neco.1997.9.8.1735.

[28] H. H. Sak, A. Senior, and B. Google, “Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling,” 2014.

[29] K. Cho *et al.*, “Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation,” 2014.

[30] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning Data Mining, Inference, and Prediction*, Second Edition. 2008.

[31] G. Cong, O. Bhardwaj, and M. Feng, “An Efficient, Distributed Stochastic Gradient Descent Algorithm for Deep-Learning Applications,” *Proceedings of the International Conference on Parallel Processing*, pp. 11–20, Sep. 2017, doi: 10.1109/ICPP.2017.10.

[32] S. Ruder, “An overview of gradient descent optimization algorithms *”, Accessed: Aug. 08, 2023. [Online]. Available: http://caffe.berkeleyvision.org/tutorial/solver.html

[33] S. Hochreiter, “The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions,” *https://doi.org/10.1142/S0218488598000094*, vol. 6, no. 2, pp. 107–116, Nov. 2011, doi: 10.1142/S0218488598000094.

[34] M. Liu, L. Chen, X. Du, L. Jin, and M. Shang, “Activated Gradients for Deep Neural Networks,” *IEEE Trans Neural Netwok Learn Syst*, vol. 34, no. 4, pp. 2156–2168, Apr. 2023, doi: 10.1109/TNNLS.2021.3106044.

[35] B. Liu, Z. Liu, T. Zhang, and T. Yuan, “Non-differentiable saddle points and sub-optimal local minima exist for deep ReLU networks,” *Neural Networks*, vol. 144, pp. 75–89, 2021, doi: 10.1016/j.neunet.2021.08.005.

[36] Y. N. Dauphin, R. Pascanu, C. Gulcehre, K. Cho, S. Ganguli, and Y. Bengio, “Identifying and attacking the saddle point problem in high-dimensional non-convex optimization,” 2014.

[37] R. Caruana, L. Pratt, and S. Thrun, “Multitask Learning,” vol. 28, pp. 41–75, 1997.

[38] R. Collobert and J. Weston, “A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning,” 2008, [Online]. Available: http://wordnet.princeton.edu

[39] S. El-Sappagh, T. Abuhmed, S. M. Riazul Islam, and K. S. Kwak, “Multimodal multitask deep learning model for Alzheimer’s disease progression detection based on time series data,” *Neurocomputing*, vol. 412, pp. 197–215, Oct. 2020, doi: 10.1016/J.NEUCOM.2020.05.087.

[40] L. Di Persio, O. Honchar, and L. DI Persio, “Multitask machine learning for financial forecasting Forward-Looking Volatility Estimation for Risk-Managed Investment Strategies after the Covid-19 Crisis View project Multitask machine learning for financial forecasting,” 2018.

[41] K. Olorunnimbe and H. Viktor, “Deep learning in the stock market—a systematic survey of practice, backtesting, and applications,” *Artif Intell Rev*, 2022, doi: 10.1007/s10462-022-10226-0.

[42] L. Zhang, C. Aggarwal, and G.-J. Qi, “Stock Price Prediction via Discovering Multi-Frequency Trading Pat-terns,” *KDD*, vol. 17, 2017, doi: 10.1145/3097983.3098117.

[43] X. Ding, Y. Zhang, T. Liu, and J. Duan, “Deep Learning for Event-Driven Stock Prediction,” *2015*.

[44] J. Shen and M. Omair Shafiq, “Short-term stock market price trend prediction using a comprehensive deep learning system,” *2020*, doi: 10.1186/s40537-020-00333-6.

[45] Y.-C. Tsai, J.-H. Chen, and J.-J. Wang, “Predict Forex Trend via Convolutional Neural Networks,” 2018.

[46] A. García-Medina, E. Aguayo-Moreno, A. García-Medina, and E. Aguayo-Moreno, “LSTM-GARCH Hybrid Model for the Prediction of Volatility in Cryptocurrency Portfolios,” 2023, doi: 10.1007/s10614-023-10373-8.

[47] G. Jung and S.-Y. Choi, “Forecasting Foreign Exchange Volatility Using Deep Learning Autoencoder-LSTM Techniques,” 2021, doi: 10.1155/2021/6647534.

[48] V. Ingle and S. Deshmukh, “Ensemble deep learning framework for stock market data prediction (EDLF-DP),” *Global Transitions Proceedings*, vol. 2, no. 1, pp. 47–66, Jun. 2021, doi: 10.1016/j.gltp.2021.01.008.

[49] Y. Li and Y. Pan, “A novel ensemble deep learning model for stock prediction based on stock prices and news,” *Int J Data Sci Anal*, vol. 13, pp. 139–149, 2022, doi: 10.1007/s41060-021-00279-9.

[50] M. Kraus and S. Feuerriegel, “Decision support from financial disclosures with deep neural networks and transfer learning,” Oct. 2017, doi: 10.1016/j.dss.2017.10.001.

[51] O. Assaf, G. Di Fatta, and G. Nicosia, “Multivariate LSTM for Stock Market Volatility Prediction,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 13164 LNCS, pp. 531–544, 2022, doi: 10.1007/978-3-030-95470-3_40/COVER.

[52] L. Yang, J. Li, R. Dong, Y. Zhang, and B. Smyth, “NumHTML: Numeric-Oriented Hierarchical Transformer Model for Multi-task Financial Forecasting,” Jan. 2022, [Online]. Available: http://arxiv.org/abs/2201.01770

[53] J. Ong and D. Herremans, “Constructing Time-Series Momentum Portfolios with Deep Multi-Task Learning,” Jun. 2023, doi: 10.1016/j.eswa.2023.120587.

[54] R. Sawhney, P. Mathur, A. Mangal, P. Khanna, R. Ratn Shah, and R. Zimmermann, “Multimodal Multi-Task Financial Risk Forecasting,” p. 10, 2020, doi: 10.1145/3394171.3413752.

[55] T. Mehmood, A. E. Gerevini, A. Lavelli, and I. Serina, “ScienceDirect Combining Multi-task Learning with Transfer Learning for Biomedical Named Entity Recognition,” 2020, doi: 10.1016/j.procs.2020.09.080.

[56] S. Il Lee, · Seong, and J. Yoo, “Multimodal deep learning for finance: integrating and forecasting international stock markets,” *J Supercomput*, vol. 76, pp. 8294–8312, 2019, doi: 10.1007/s11227-019-03101-3.

[57] C. Yuan, X. Ma, H. Wang, C. Zhang, and X. Li, “COVID19-MLSF: A multi-task learning-based stock market forecasting framework during the COVID-19 pandemic,” *Expert Syst Appl*, vol. 217, p. 119549, 2023, doi: 10.1016/j.eswa.2023.119549.

[58] Fu Kai and Xu Wenhua, “Training neural network with genetic algorithms for forecasting the stock price index,” in *1997 IEEE International Conference on Intelligent Processing Systems (Cat. No.97TH8335)*, IEEE, pp. 401–403. doi: 10.1109/ICIPS.1997.672809.

[59] G. Dorffner, “Neural Computation and Applications in Time Series and Signal Processing,” *J Signal Process Syst*, 1996.

[60] R. J. Frank, N. Davey, and S. P. Hunt, “Time Series Prediction and Neural Networks,” *J Intell Robot Syst*, vol. 31, pp. 91–103, 2001.

[61] M. Qiu, Y. Song, and F. Akagi, “Application of artificial neural network for the prediction of stock market returns: The case of the Japanese stock market,” *Chaos Solitons Fractals*, vol. 85, pp. 1–7, 2016, doi: 10.1016/j.chaos.2016.01.004.

[62] J. Z. Wang, J. J. Wang, Z. G. Zhang, and S. P. Guo, “Forecasting stock indices with back propagation neural network,” *Expert Syst Appl*, vol. 38, no. 11, pp. 14346–14355, 2011, doi: 10.1016/j.eswa.2011.04.222.

[63] T. Fischer and C. Krauss, “Deep learning with long short-term memory networks for financial market predictions,” *Eur J Oper Res*, vol. 270, no. 2, pp. 654–669, Oct. 2018, doi: 10.1016/j.ejor.2017.11.054.

[64] J. Cao, Z. Li, and J. Li, “Financial time series forecasting model based on CEEMDAN and LSTM,” *Physica A: Statistical Mechanics and its Applications*, vol. 519, pp. 127–139, Apr. 2019, doi: 10.1016/J.PHYSA.2018.11.061.

[65] Y. Baek and H. Y. Kim, “A new forecasting framework for stock market index value with an overfitting prevention LSTM module and a prediction LSTM module,” *Expert Syst Appl*, vol. 113, pp. 457–480, Dec. 2018, doi: 10.1016/J.ESWA.2018.07.019.

[66] A. Sagheer and M. Kotb, “Time series forecasting of petroleum production using deep LSTM recurrent networks,” *Neurocomputing*, vol. 323, pp. 203–213, Jan. 2019, doi: 10.1016/J.NEUCOM.2018.09.082.

[67] S. Krstanovic and H. Paulheim, “Stacked LSTM Snapshot Ensembles for Time Series Forecasting,” 2019, pp. 87–98. doi: 10.1007/978-3-030-26036-1_7.

[68] A. Sagheer and M. Kotb, “Unsupervised Pre-training of a Deep LSTM-based Stacked Autoencoder for Multivariate Time Series Forecasting Problems,” *Sci Rep*, vol. 9, no. 1, 2019, doi: 10.1038/s41598-019-55320-6.

[69] Y. Hu, J. Ni, and L. Wen, “A hybrid deep learning approach by integrating LSTM-ANN networks with GARCH model for copper price volatility prediction,” *Physica A:*

Statistical Mechanics and its Applications, vol. 557, Nov. 2020, doi: 10.1016/j.physa.2020.124907.

[70] A. Vidal and W. Kristjanpoller, “Gold volatility prediction using a CNN-LSTM approach,” *Expert Syst Appl*, vol. 157, p. 113481, Nov. 2020, doi: 10.1016/j.eswa.2020.113481.

[71] G. meng Chen, M. Firth, and O. M. Rui, “The Dynamic Relation Between Stock Returns, Trading Volume, and Volatility,” *Financial Review*, vol. 36, no. 3, pp. 153–174, Aug. 2001, doi: 10.1111/J.1540-6288.2001.TB00024.X.

[72] A. F. Darrat, S. Rahman, and M. Zhong, “Intraday trading volume and return volatility of the DJIA stocks: A note,” *J Bank Financ*, vol. 27, no. 10, pp. 2035–2043, Oct. 2003, doi: 10.1016/S0378-4266(02)00321-7.

[73] A. Kudryavtsev, “The effect of stock return sequences on trading volumes,” *International Journal of Financial Studies*, vol. 5, no. 4, pp. 1–15, 2017, doi: 10.3390/ijfs5040020.

[74] D. Lawa Tan Toe and · Salifou Ouedraogo, “Dynamic relationship between trading volume, returns and returns volatility: an empirical investigation on the main African’s stock markets,” *Journal of Asset Management*, 2022, doi: 10.1057/s41260-022-00274-0.

[75] G. Connor, “Active Portfolio Management: A Quantitative Approach to Providing Superior Returns and Controlling Risk,” *Review of Financial Studies*, vol. 13, no. 4, 2000, doi: 10.1093/rfs/13.4.1153.

[76] R. Engle, “GARCH 101: The Use of ARCH/GARCH Models in Applied Econometrics,” *Journal of Economic Perspectives*, vol. 15, no. 4, pp. 157–168, 2001, doi: 10.1257/JEP.15.4.157.

[77] J. B. Heaton, N. G. Polson, and J. H. Witte, “Deep learning for finance: deep portfolios,” *Appl Stoch Models Bus Ind*, vol. 33, no. 1, pp. 3–12, Jan. 2017, doi: 10.1002/ASMB.2209.

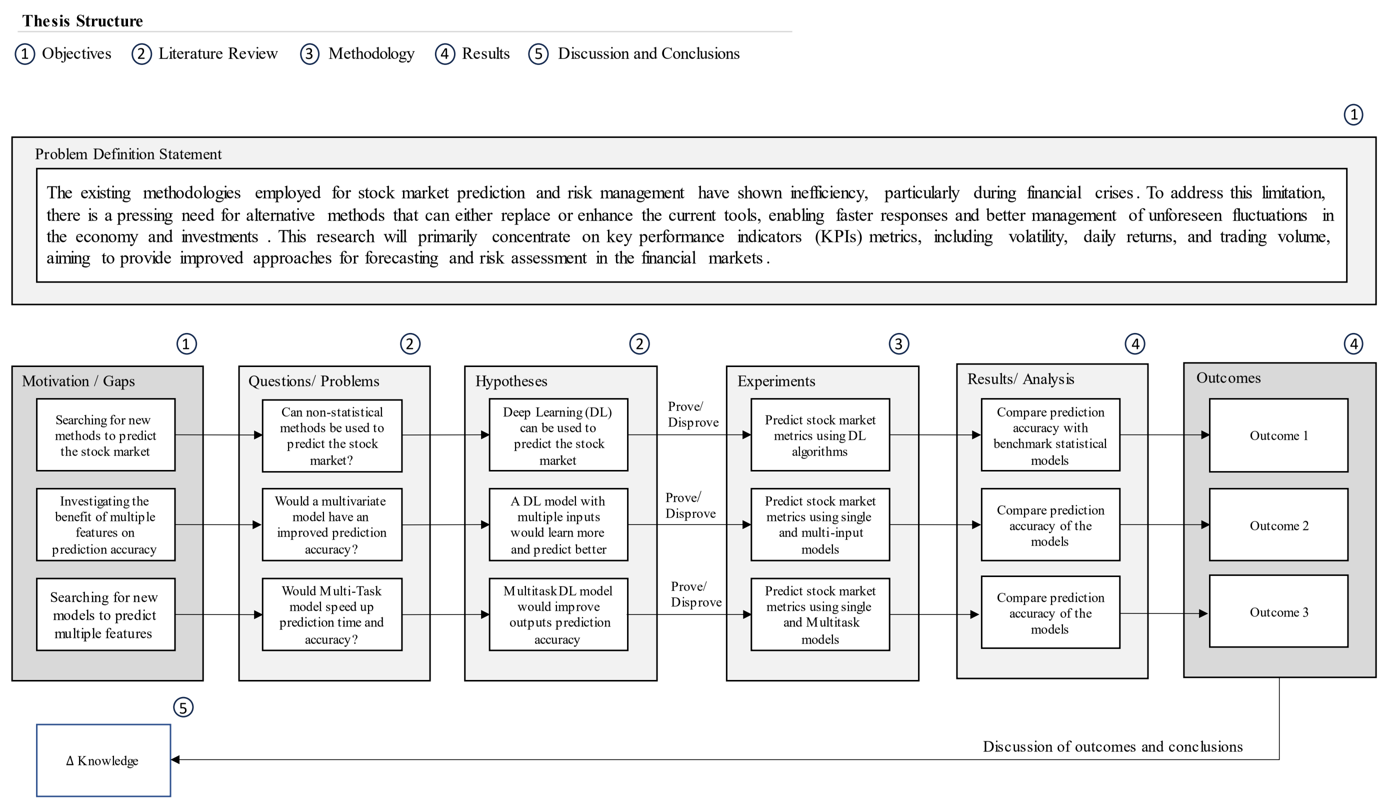
[78] A. Bhavani, A. V. Ramana, and A. S. N. Chakravarthy, “Comparative Analysis between LSTM and GRU in Stock Price Prediction,” in *International Conference on Edge Computing and Applications, ICECAA 2022 - Proceedings*, Institute of Electrical

and Electronics Engineers Inc., 2022, pp. 532–537. doi: 10.1109/ICECAA55415.2022.9936434.

[79] M. J. Hamayel and A. Y. Owda, “A Novel Cryptocurrency Price Prediction Model Using GRU, LSTM and bi-LSTM Machine Learning Algorithms,” *AI*, vol. 2, no. 4, pp. 477–496, Oct. 2021, doi: 10.3390/ai2040030.

Appendix A
Diagrams, Figures, and Charts

A.1 Diagrams


Thesis Structure
① Objectives ② Literature Review ③ Methodology ④ Results ⑤ Discussion and Conclusions
①
Problem Definition Statement
①
Motivation / Gaps
Searching for new methods to predict the stock market
Investigating the benefit of multiple features on prediction accuracy
Searching for new models to predict multiple features
②
Questions/ Problems
Can non-statistical methods be used to predict the stock market?
Would a multivariate model have an improved prediction accuracy?
Would Multi-Task model speed up prediction time and accuracy?
②
Hypotheses
Deep Learning (DL) can be used to predict the stock market
A DL model with multiple inputs would learn more and predict better
Multitask DL model would improve outputs prediction accuracy
Prove/ Disprove
Prove/ Disprove
Prove/ Disprove
③
Experiments
Predict stock market metrics using DL algorithms
Predict stock market metrics using single and multi-input models
Predict stock market metrics using single and Multitask models
④
Results/ Analysis
Compare prediction accuracy with benchmark statistical models
Compare prediction accuracy of the models
Compare prediction accuracy of the models
④
Outcomes
Outcome 1
Outcome 2
Outcome 3
Discussion of outcomes and conclusions
⑤
Δ Knowledge

The existing methodologies employed for stock market prediction and risk management have shown inefficiency, particularly during financial crises. To address this limitation, there is a pressing need for alternative methods that can either replace or enhance the current tools, enabling faster responses and better management of unforeseen fluctuations in the economy and investments . This research will primarily concentrate on key performance indicators (KPIs) metrics, including volatility, daily returns, and trading volume, aiming to provide improved approaches for forecasting and risk assessment in the financial markets .

Diagram 1: PhD project diagram.

A.2 Figures and Charts

A.2.1 Multivariate Deep Learning – Chapter 5

A.2.1.1 Benchmark Testing

A.2.1.1.1 GARCH Model

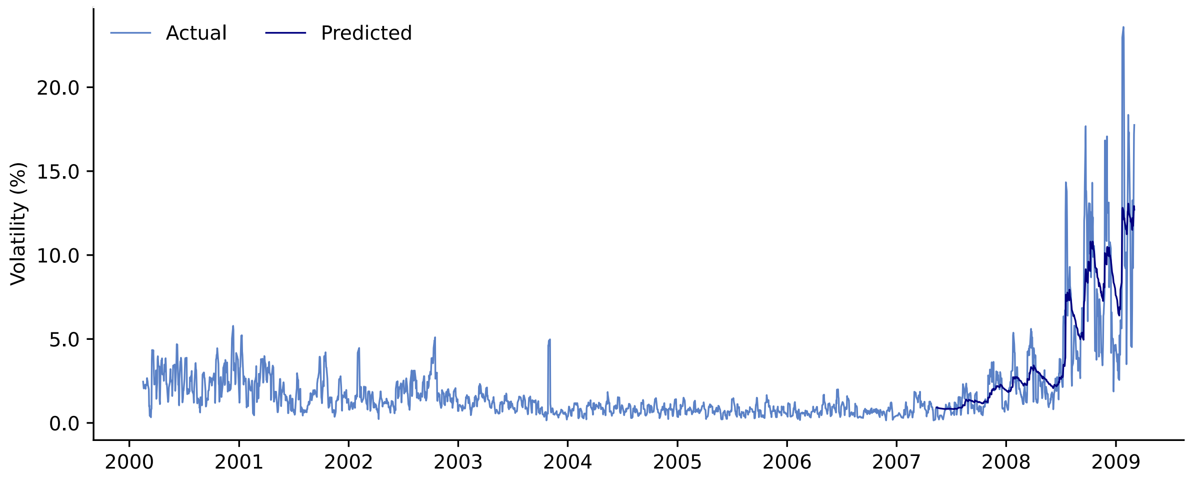


Fig 1: BAC price volatility prediction in bear market (long window).

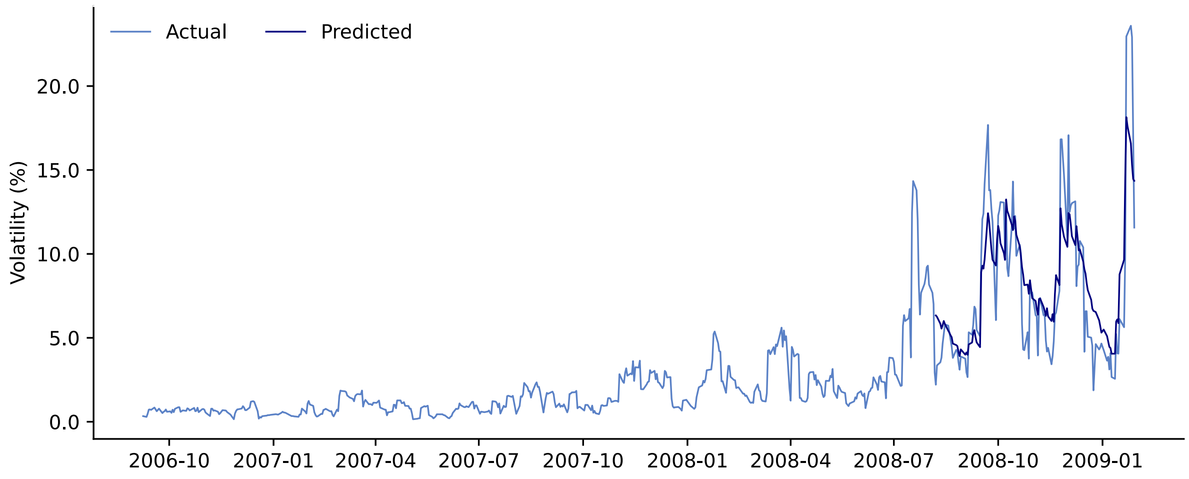


Fig 2: BAC price volatility prediction in bear market (short window).

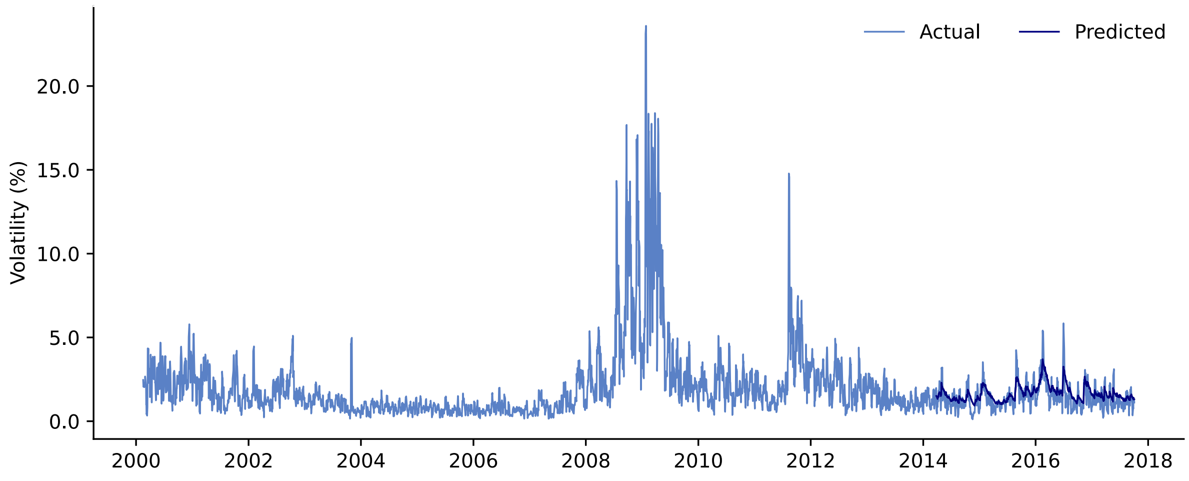


Fig 3: BAC price volatility prediction in bull market (long window).

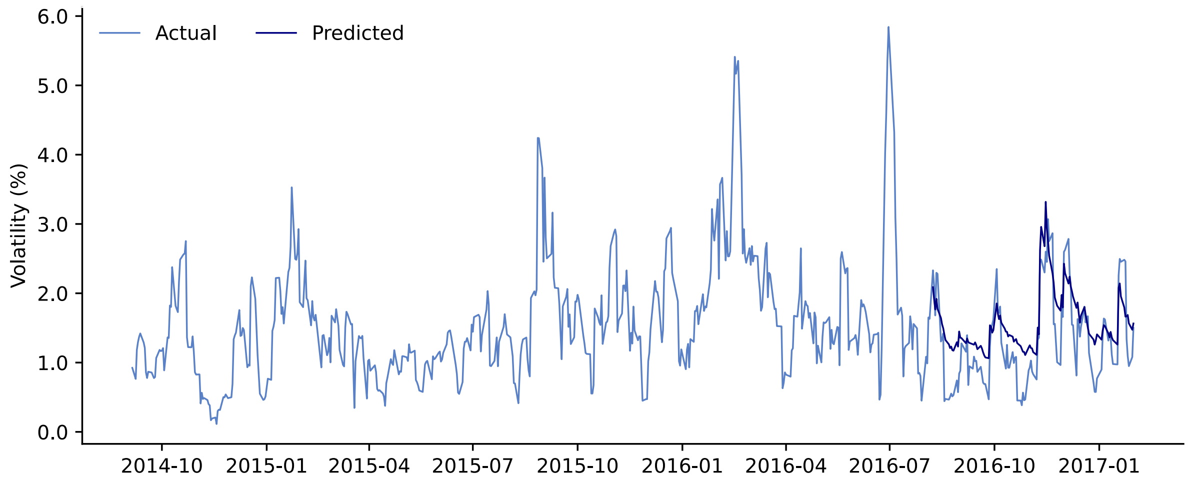


Fig 4: BAC price volatility prediction in bull market (short window).

A.2.1.1.2 Deep Learning Model (LSTM)

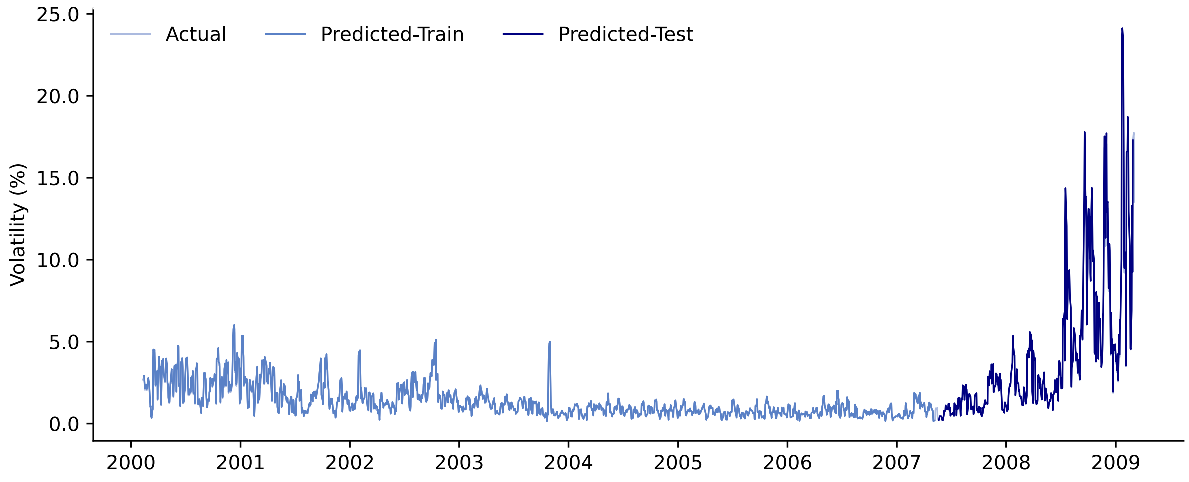


Fig 5: BAC price volatility prediction in bear market (long window).

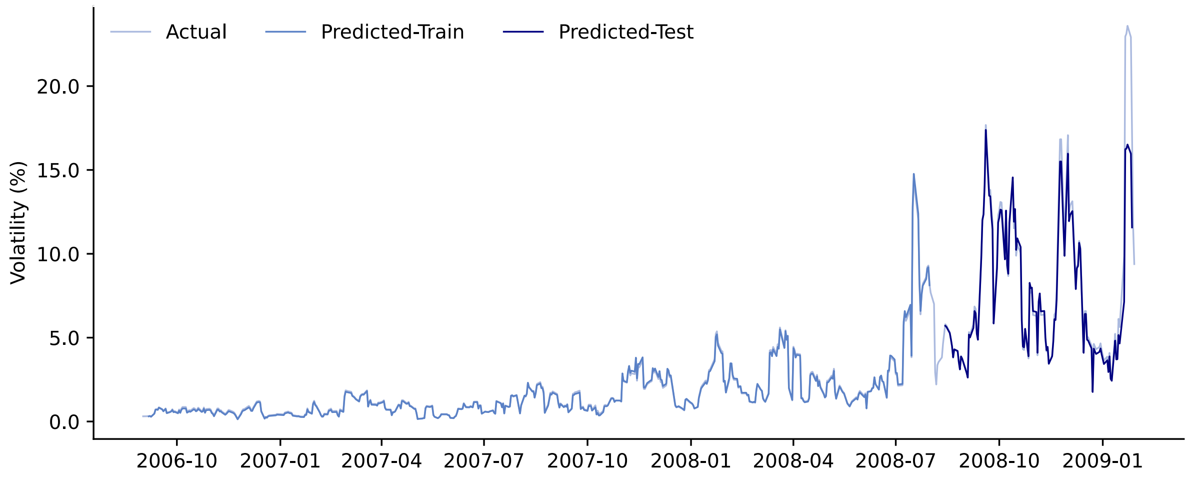


Fig 6: BAC stock price volatility prediction in bear market (short window).

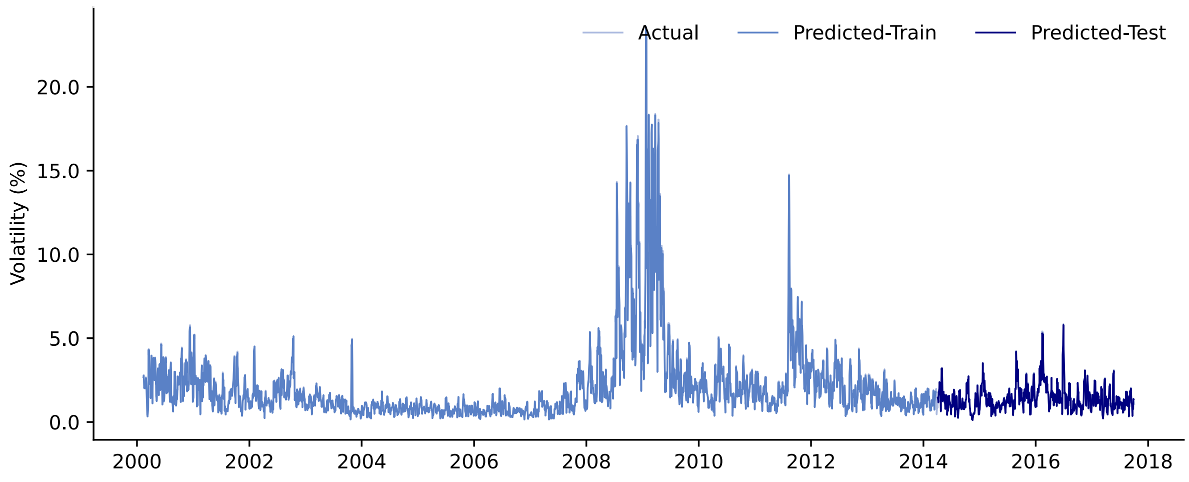


Fig 7: BAC price volatility prediction in bull market (long window).

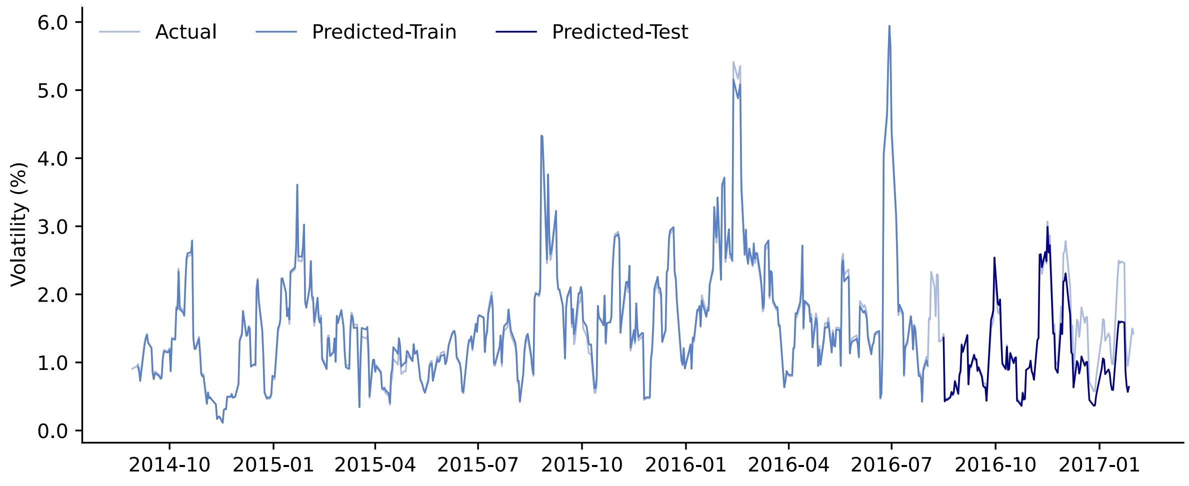


Fig 8: BAC price volatility prediction in bull market (short window).

A.2.1.2 Multivariate vs Single-Input Deep Learning Models

A.2.1.2.1 Single-Input Deep Learning Model

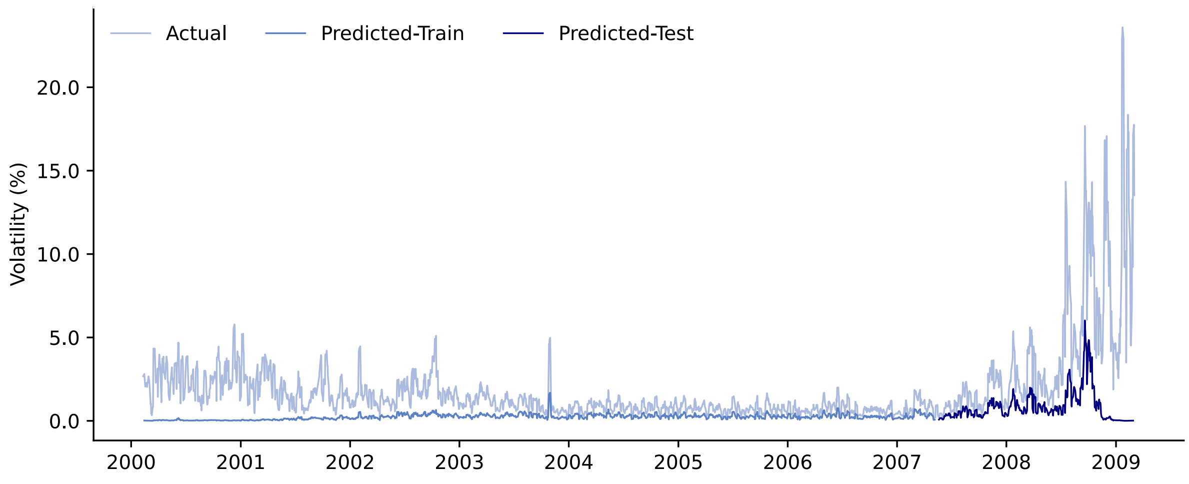


Fig 9: BAC price volatility prediction in bear market.

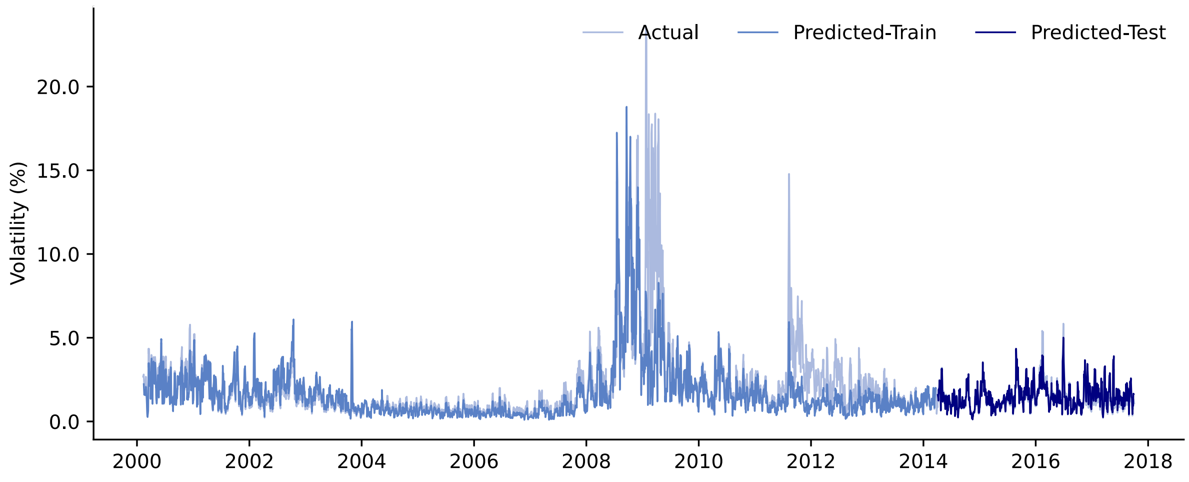


Fig 10: BAC price volatility prediction in bull market.

A.1.1.2.2 Multivariate Deep Learning Model

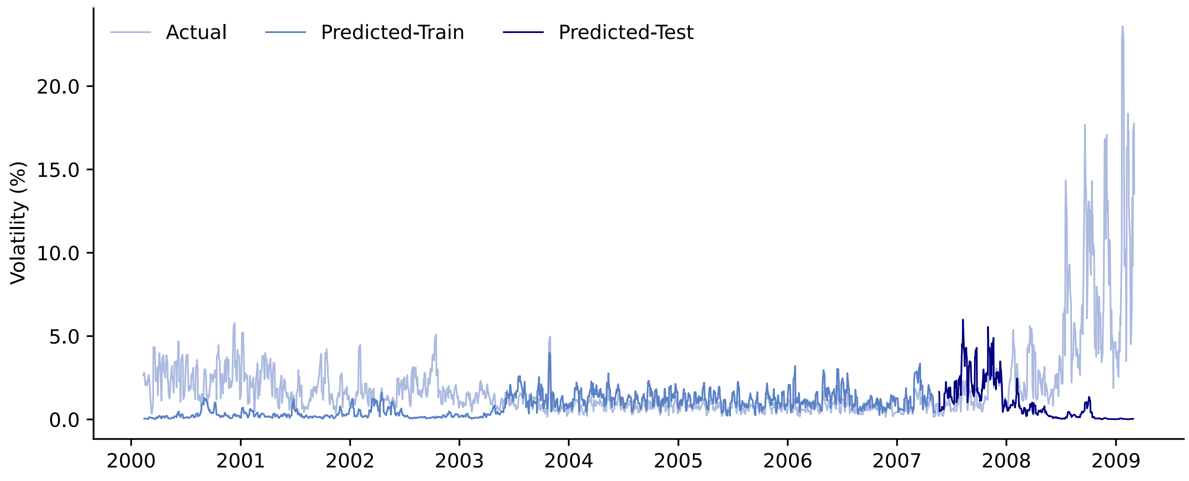


Fig 11: BAC price volatility prediction in bear market with C price feature input.

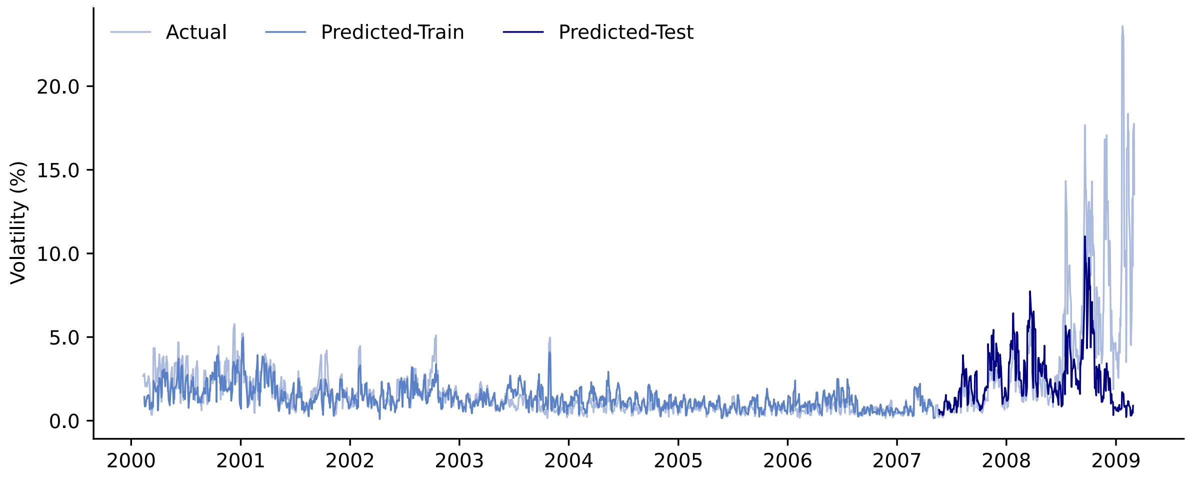


Fig 12: BAC price volatility prediction in bear market with DB and C price feature inputs.

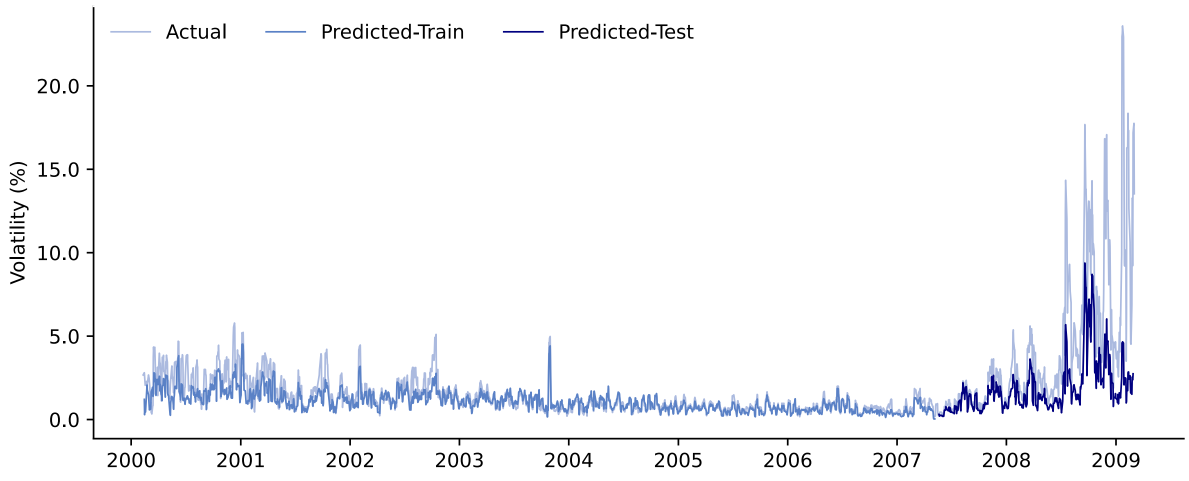


Fig 13: BAC price volatility prediction in bear market with DB, GS, and C and price feature inputs.

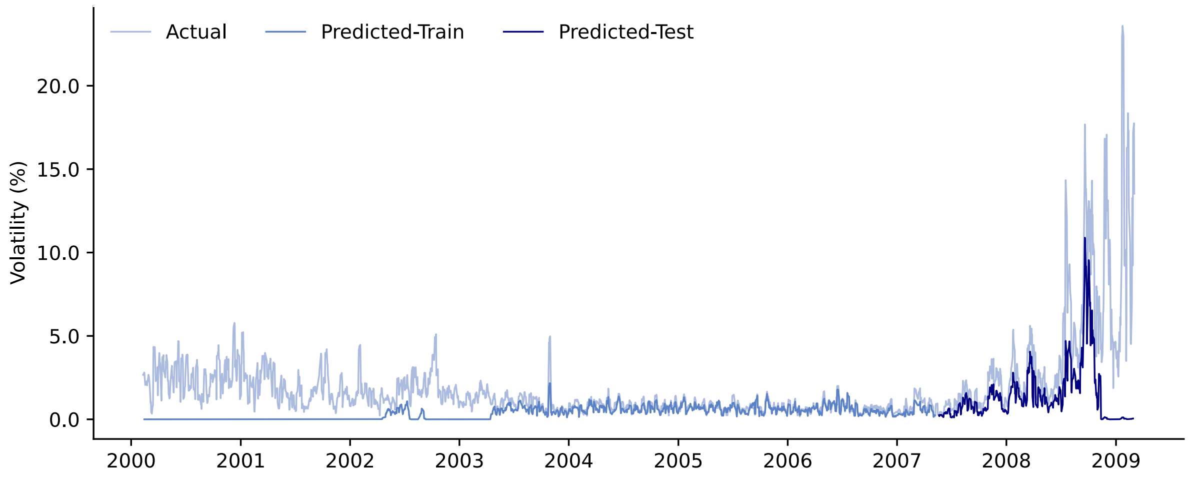


Fig 14: BAC price volatility prediction in bear market with DB, GS, C, and MS price feature inputs.

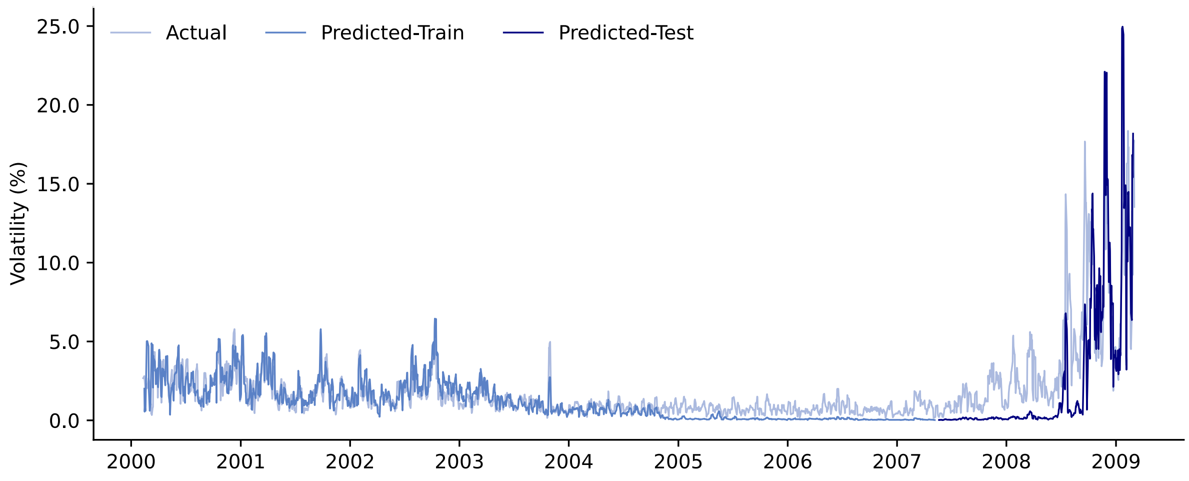


Fig 15: BAC price volatility prediction in bear market with DB, GS price feature inputs.

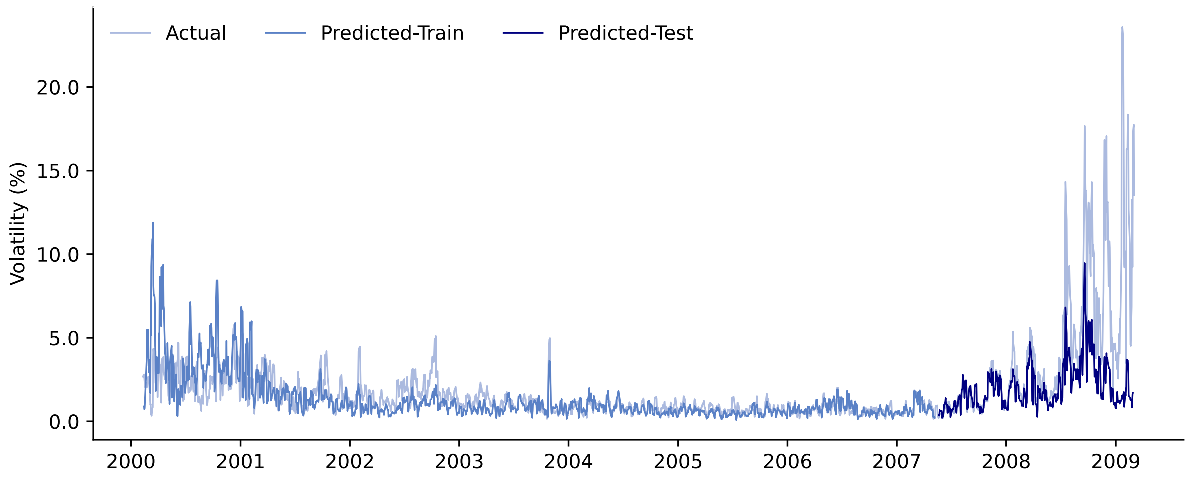


Fig 16: BAC price volatility prediction in bear market with DB and MS price feature inputs.

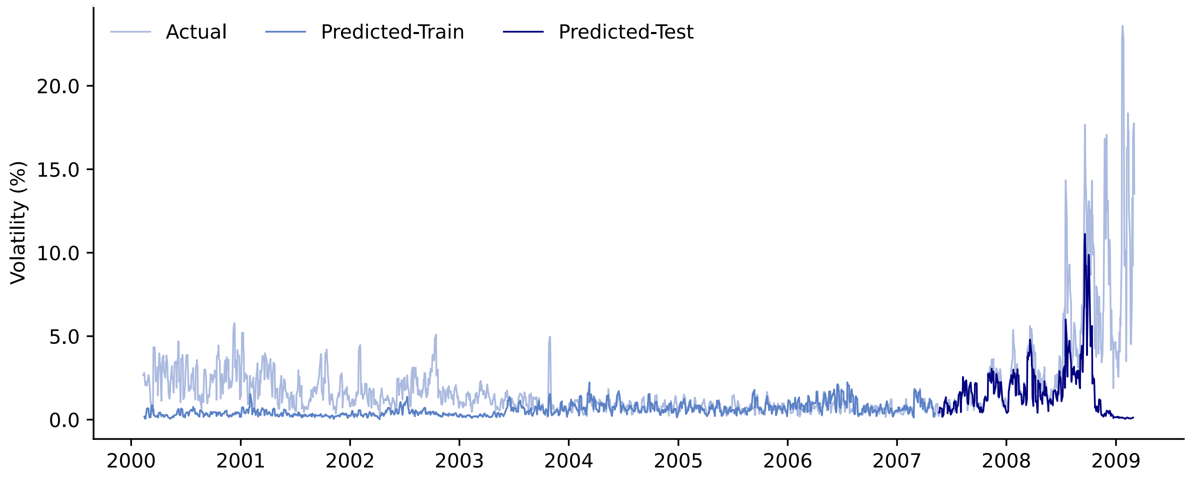


Fig 17: BAC price volatility prediction in bear market with DB price feature input

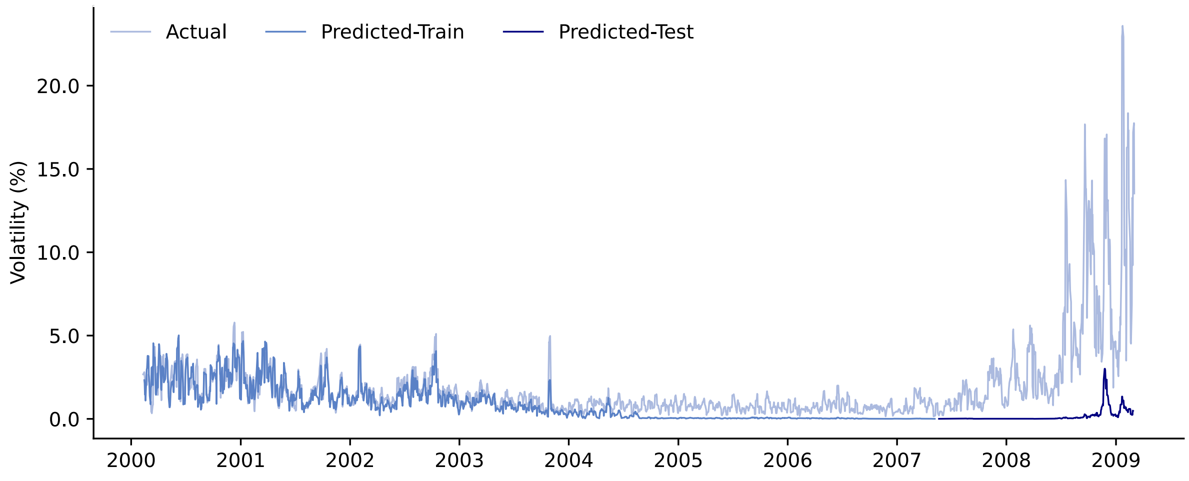


Fig 18: BAC price volatility prediction in bear market with GOLD and DB price feature inputs.

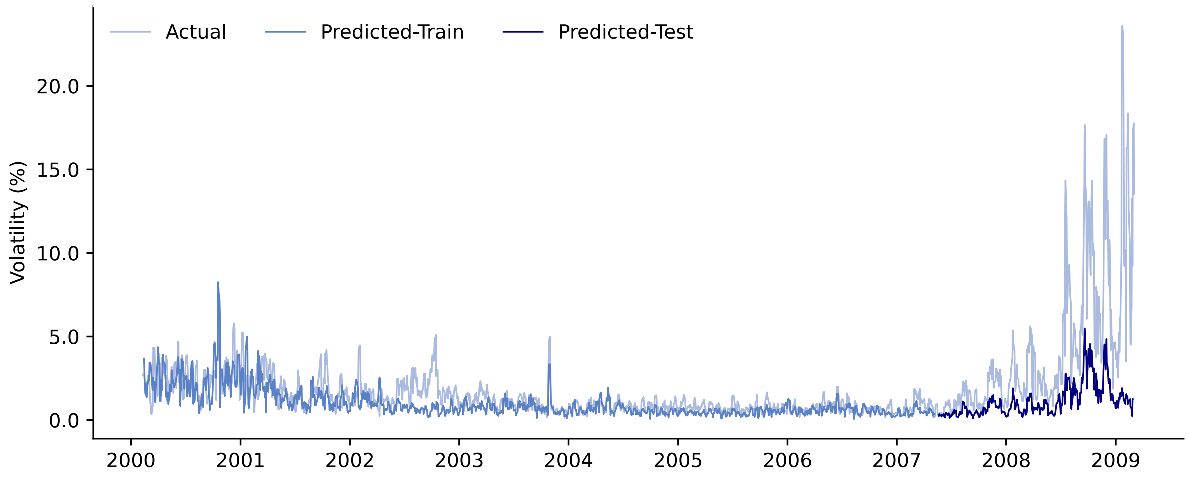


Fig 19: BAC price volatility prediction in bear market with GOLD and IBM price feature inputs.

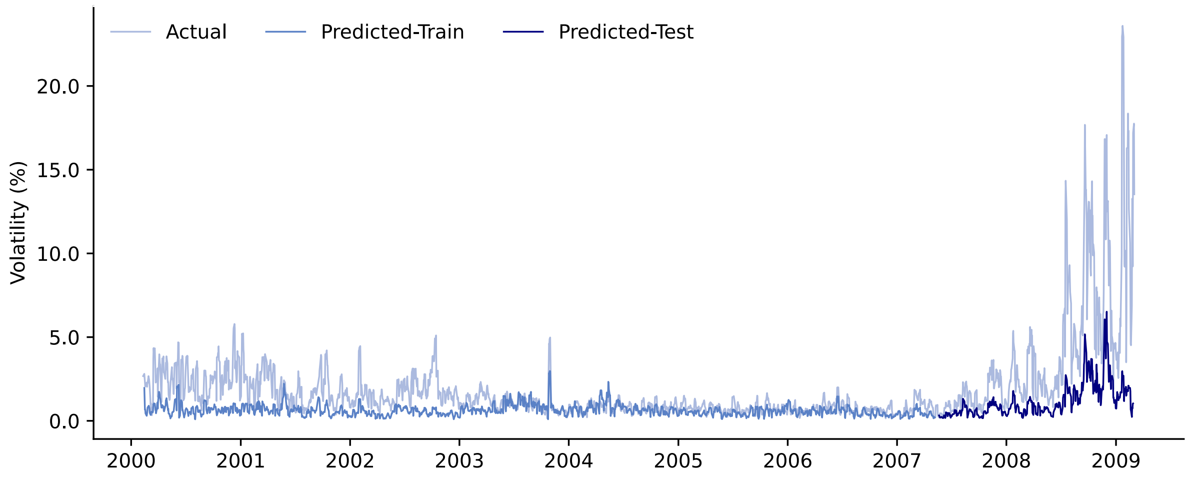


Fig 20: BAC price volatility prediction in bear market with GOLD and JPM price feature inputs.

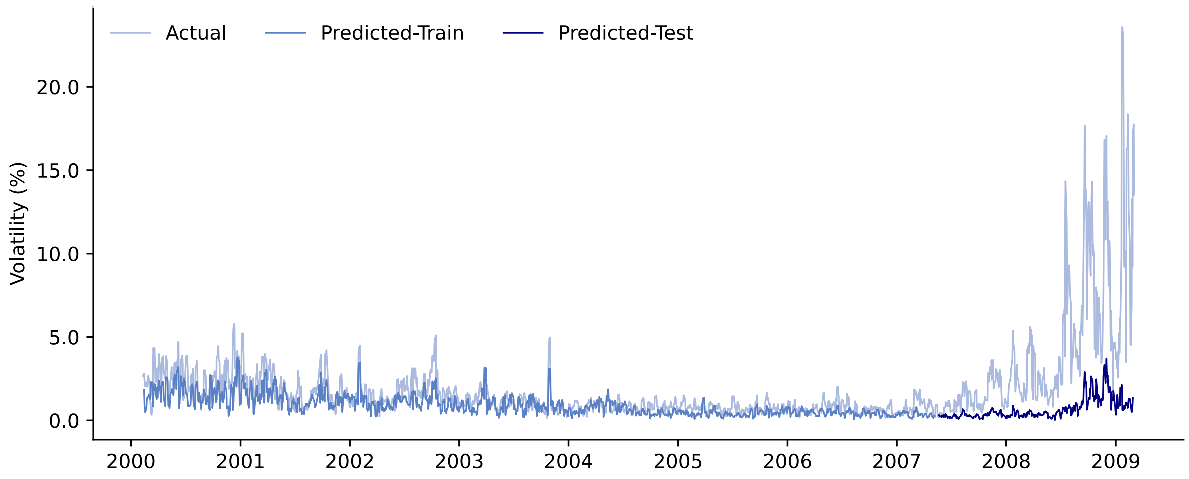


Fig 21: BAC price volatility prediction in bear market with GOLD and OIL price feature inputs.

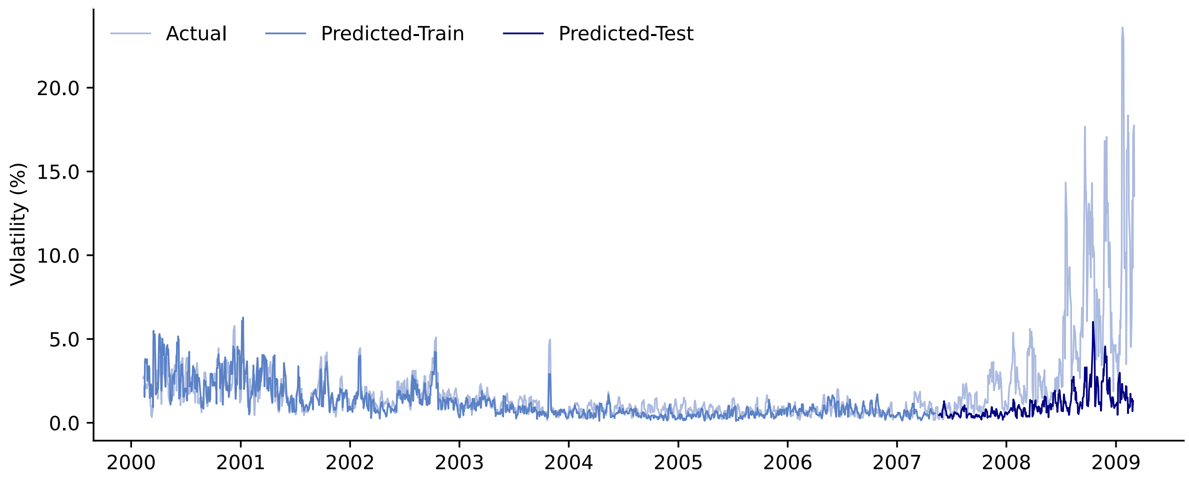


Fig 22: BAC price volatility prediction in bear market with GOLD and WMT price feature inputs.

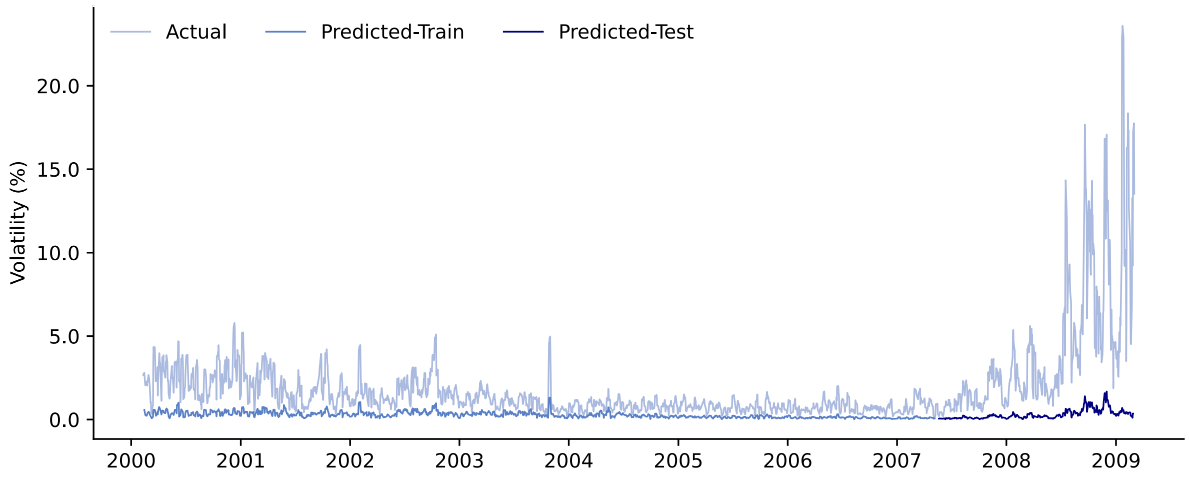


Fig 23: BAC price volatility prediction in bear market with GOLD price feature input.

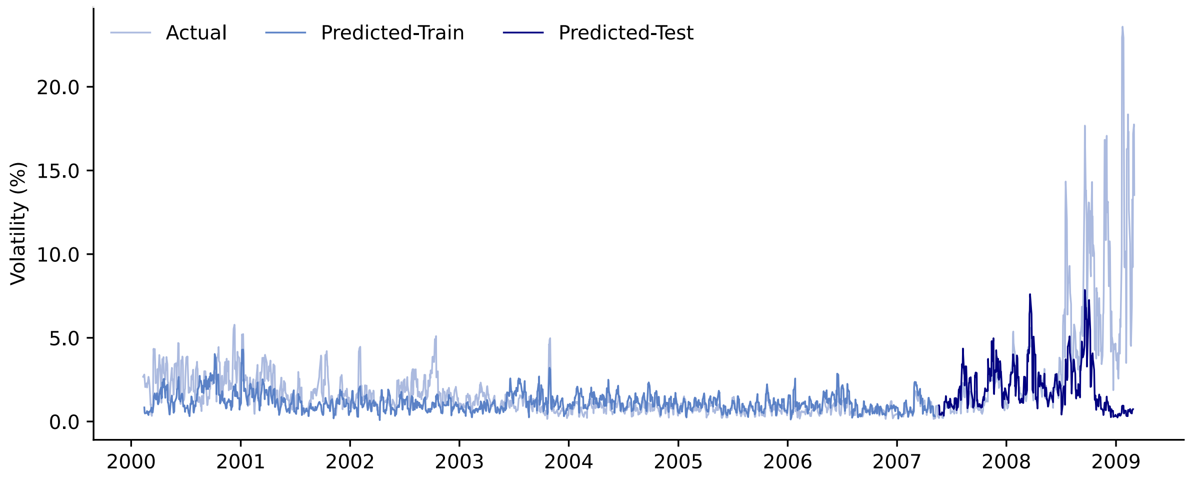


Fig 24: BAC price volatility prediction in bear market with GS and C price feature inputs.

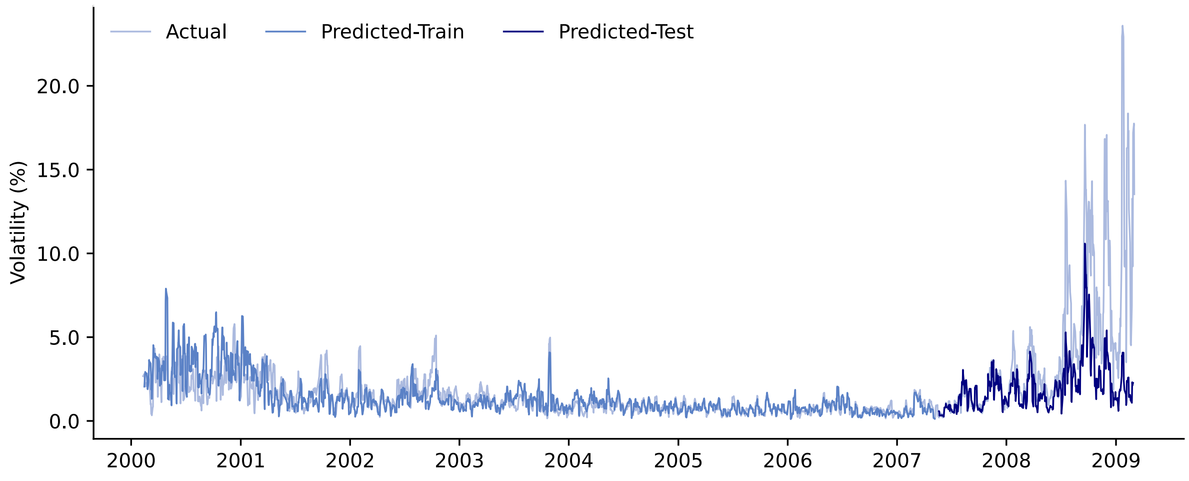


Fig 25: BAC price volatility prediction in bear market with GS, MS, and C price feature inputs.

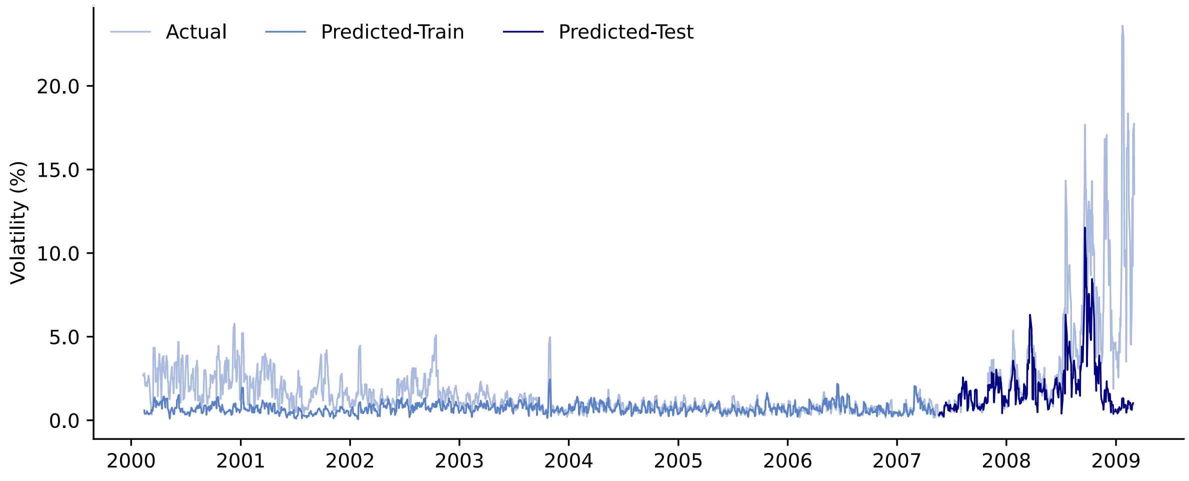


Fig 26: BAC price volatility prediction in bear market with GS and MS price feature inputs.

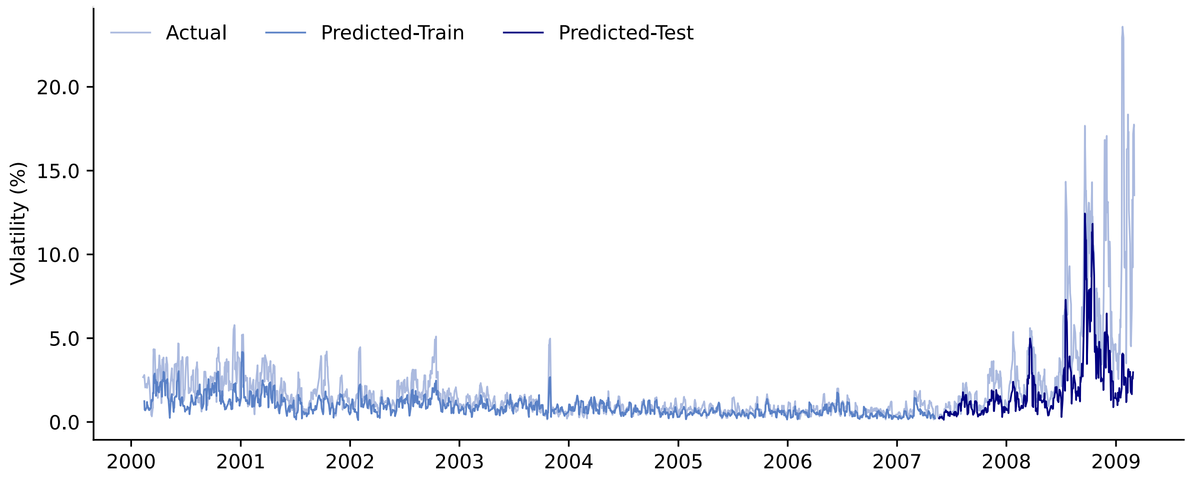


Fig 27: BAC price volatility prediction in bear market with GS price feature inputs.

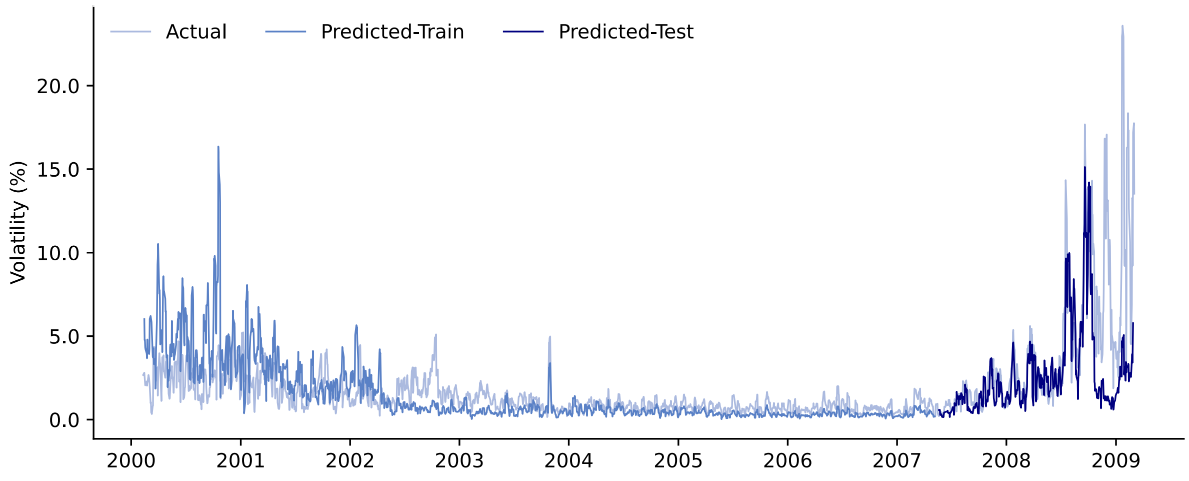


Fig 28: BAC price volatility prediction in bear market with IBM price feature input.

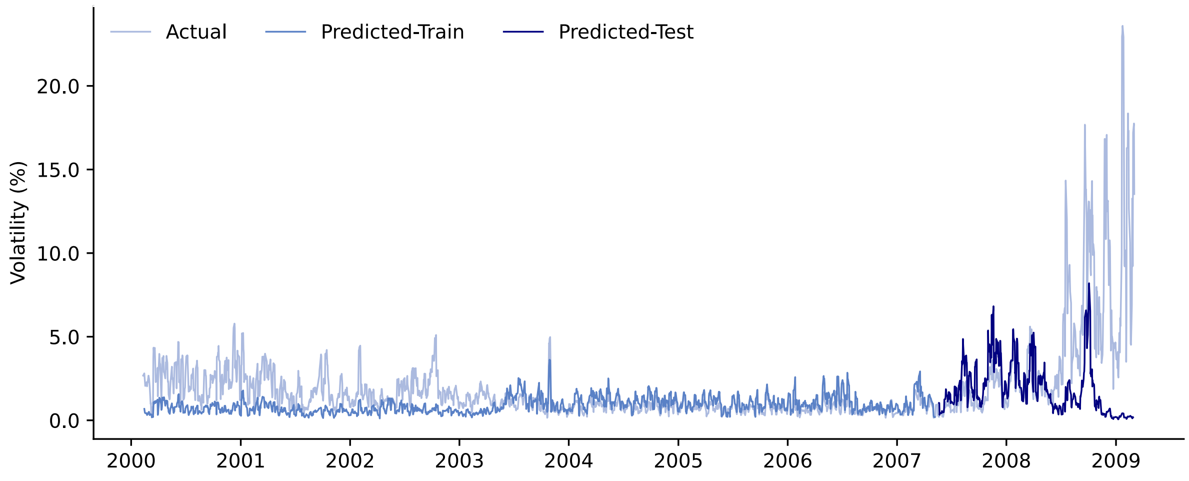


Fig 29: BAC price volatility prediction in bear market with JPM and C price feature inputs.

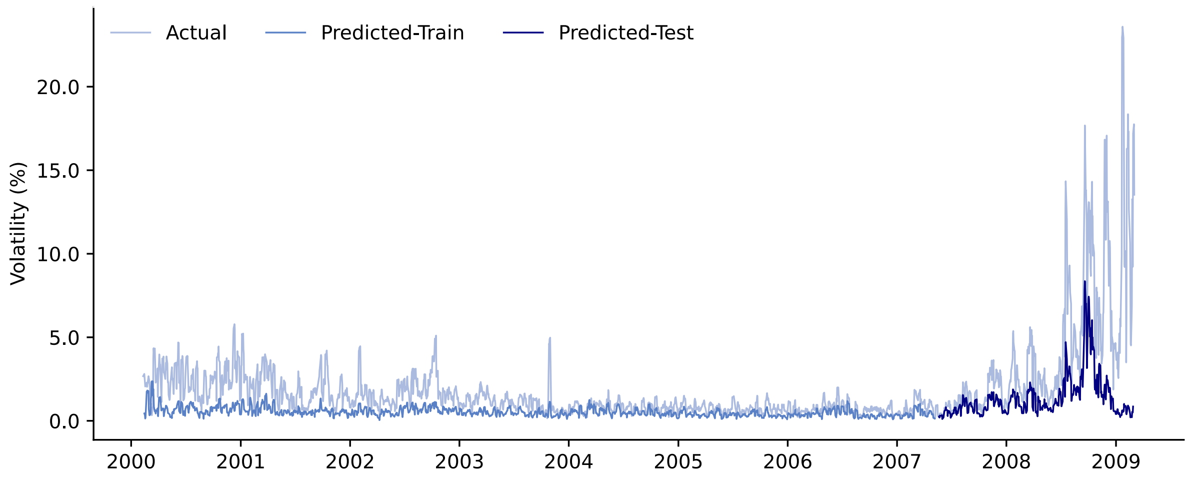


Fig 39: BAC price volatility prediction in bear market with JPM and DB price feature inputs.

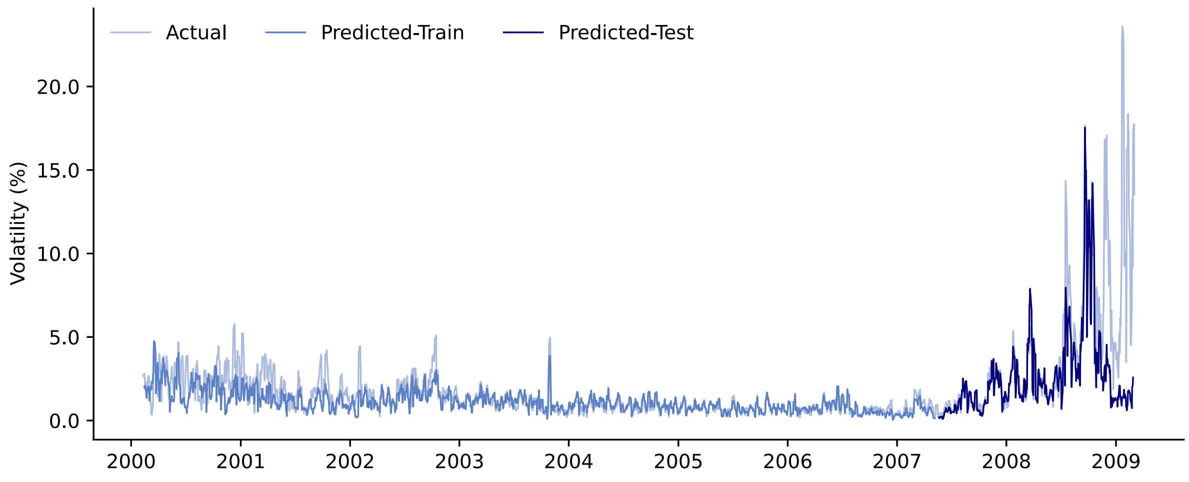


Fig 31: BAC price volatility prediction in bear market with JPM, GS and C price feature inputs.

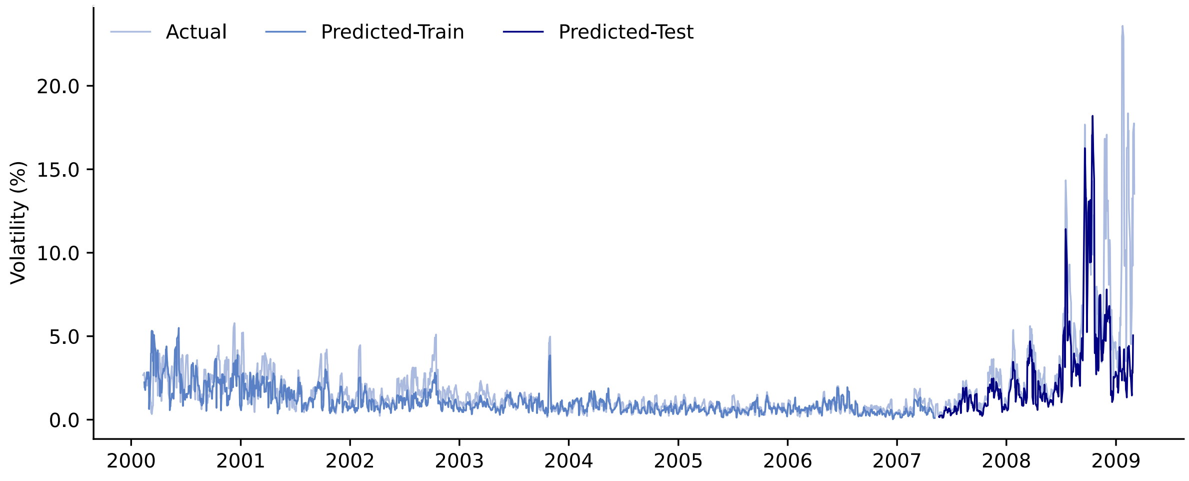


Fig 32: BAC price volatility prediction in bear market with JPM, GS and DB price feature inputs.

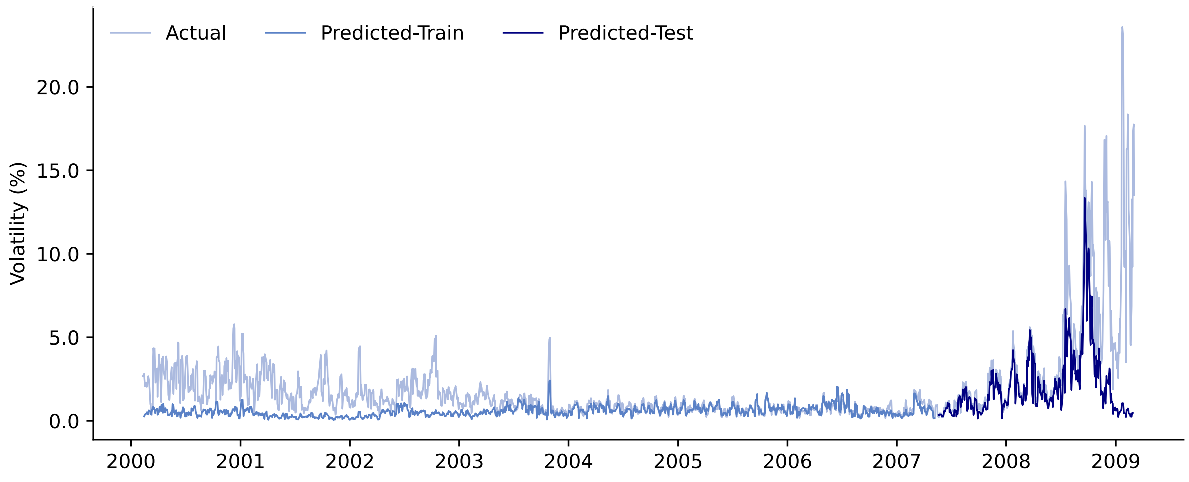


Fig 33: BAC price volatility prediction in bear market with JPM, GS, and MS price feature inputs.

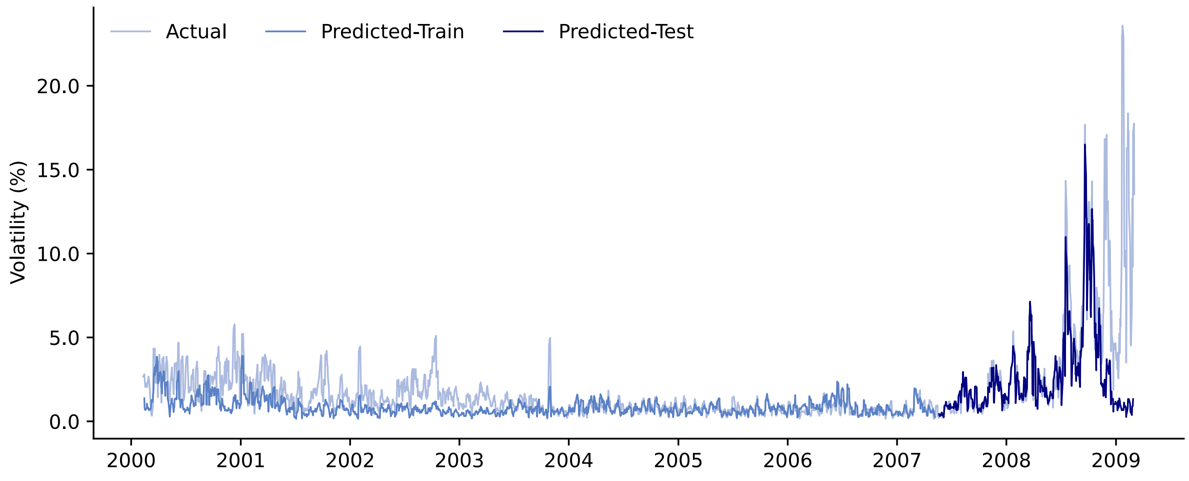


Fig 34: BAC price volatility prediction in bear market with JPM and GS price feature inputs.

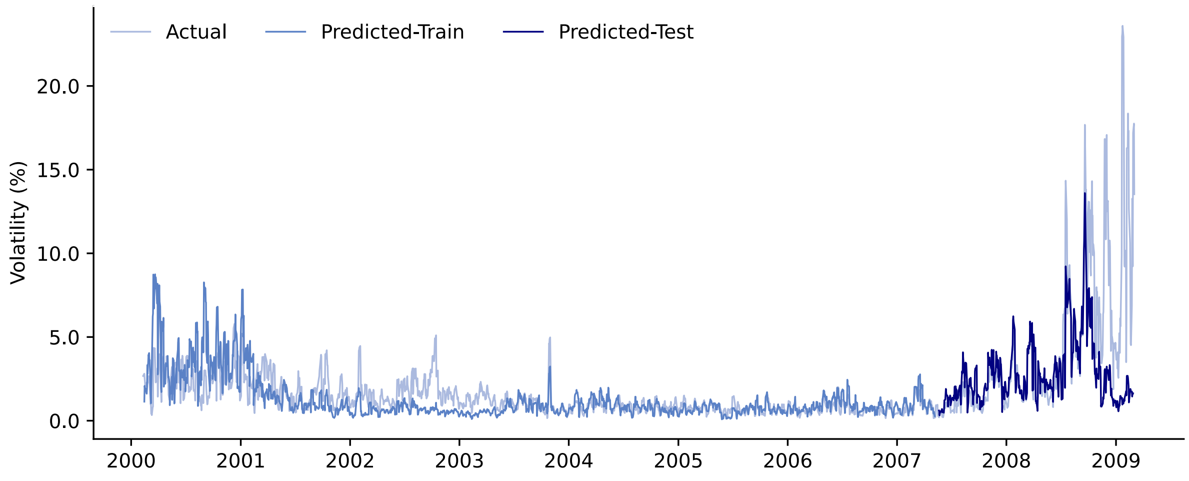


Fig 35: BAC price volatility prediction in bear market with JPM and MS price feature inputs.

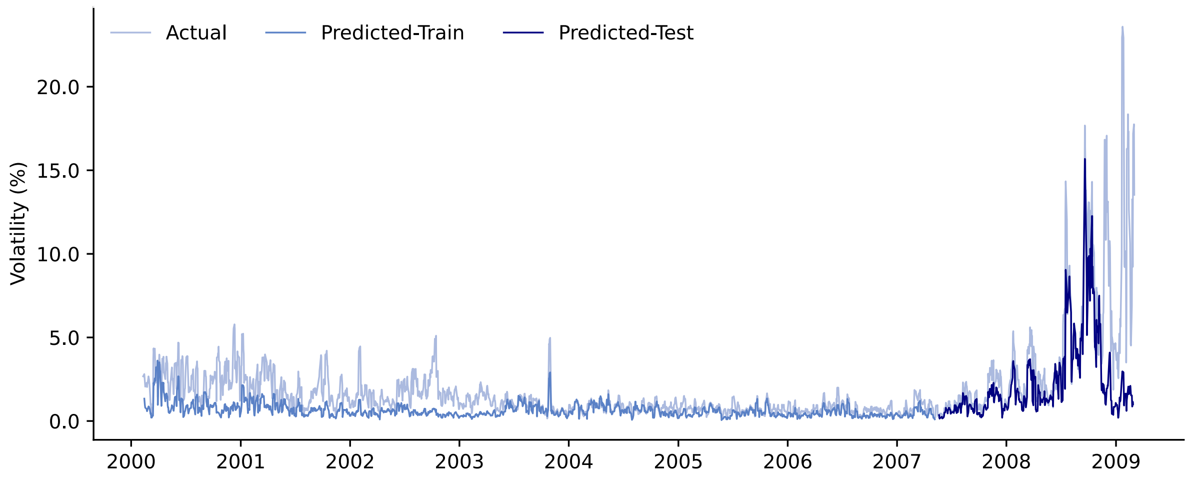


Fig 36: BAC price volatility prediction in bear market with JPM price feature input.

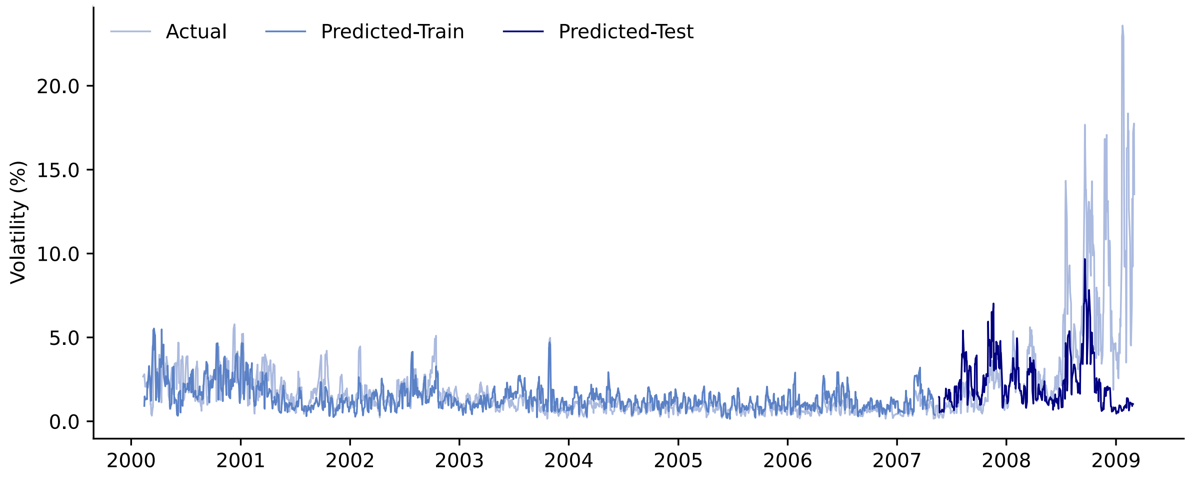


Fig 37: BAC price volatility prediction in bear market with MS and C price feature inputs.

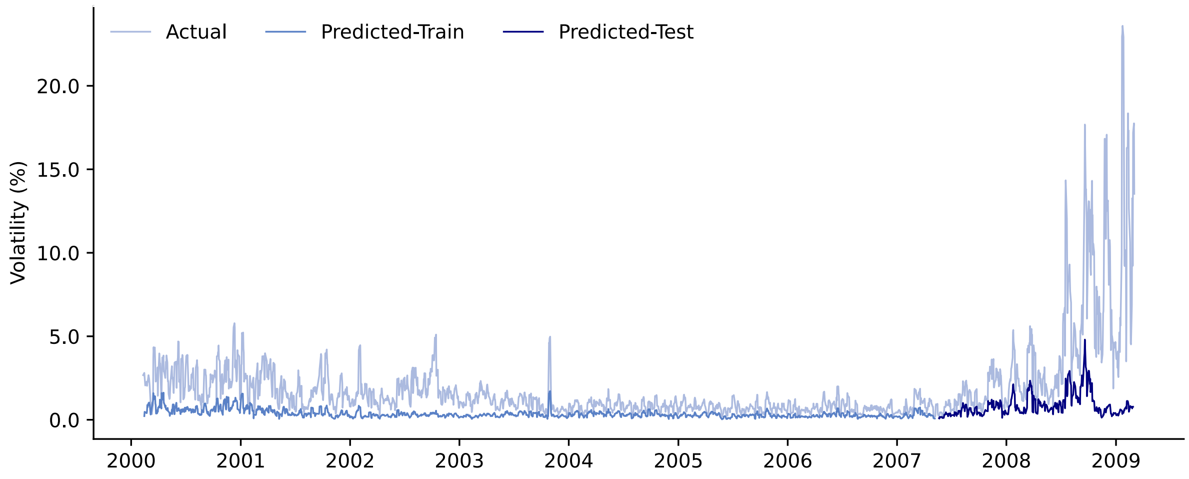


Fig 38: BAC price volatility prediction in bear market with MS price feature input.

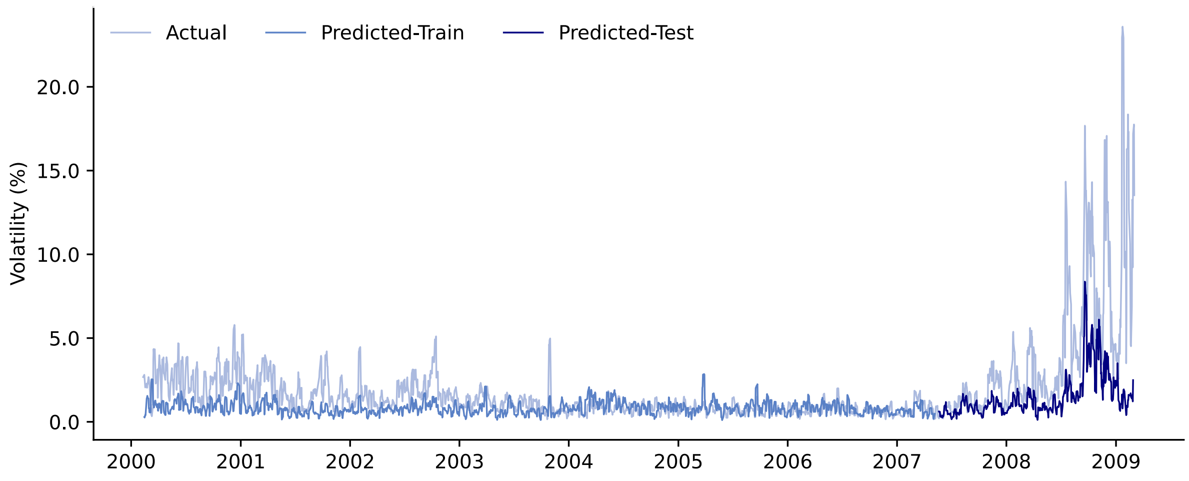


Fig 39: BAC price volatility prediction in bear market with OIL and DB price feature inputs.

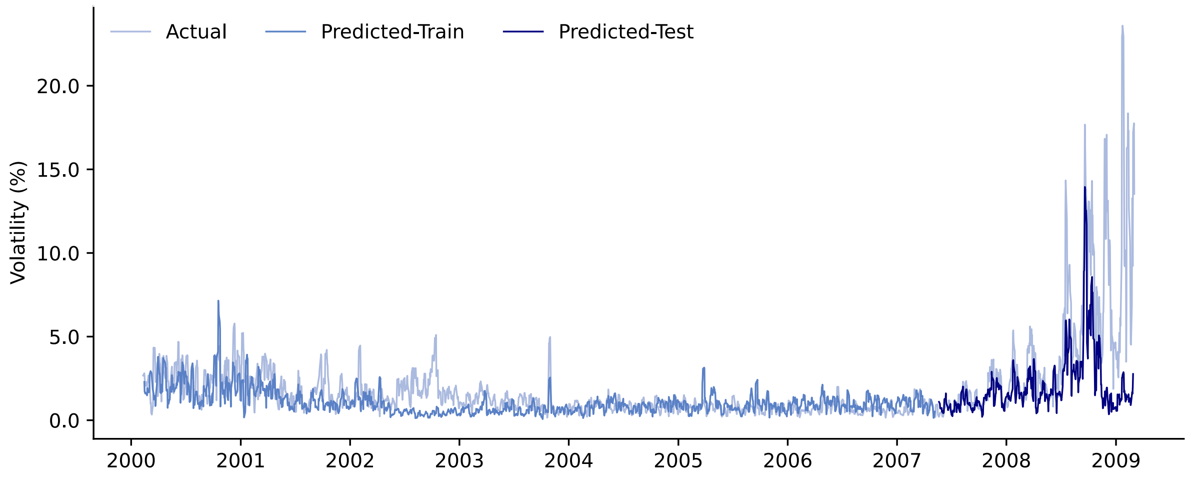


Fig 40: BAC price volatility prediction in bear market with OIL and IBM price feature inputs.

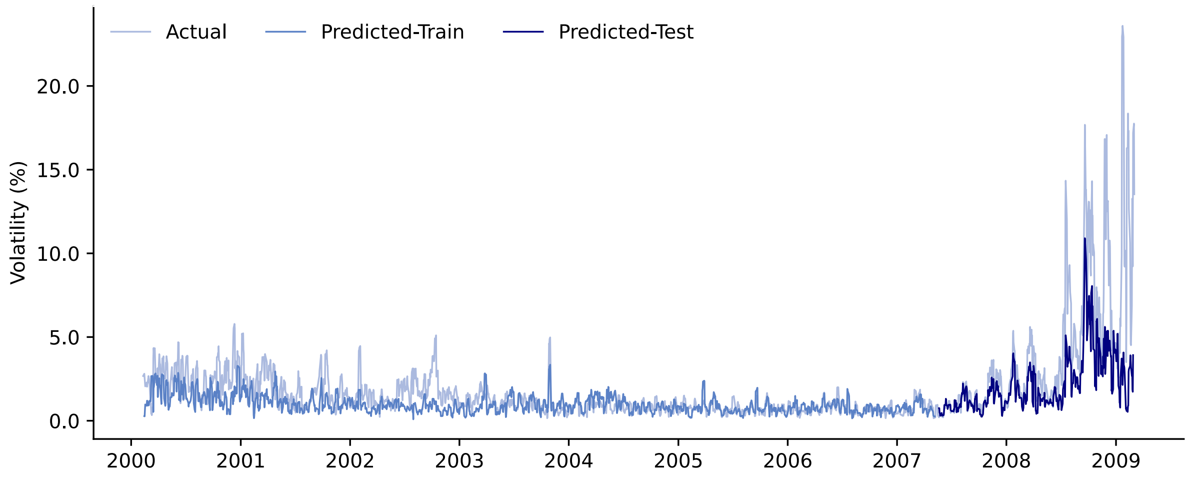


Fig 41: BAC price volatility prediction in bear market with OIL and JPM price feature inputs.

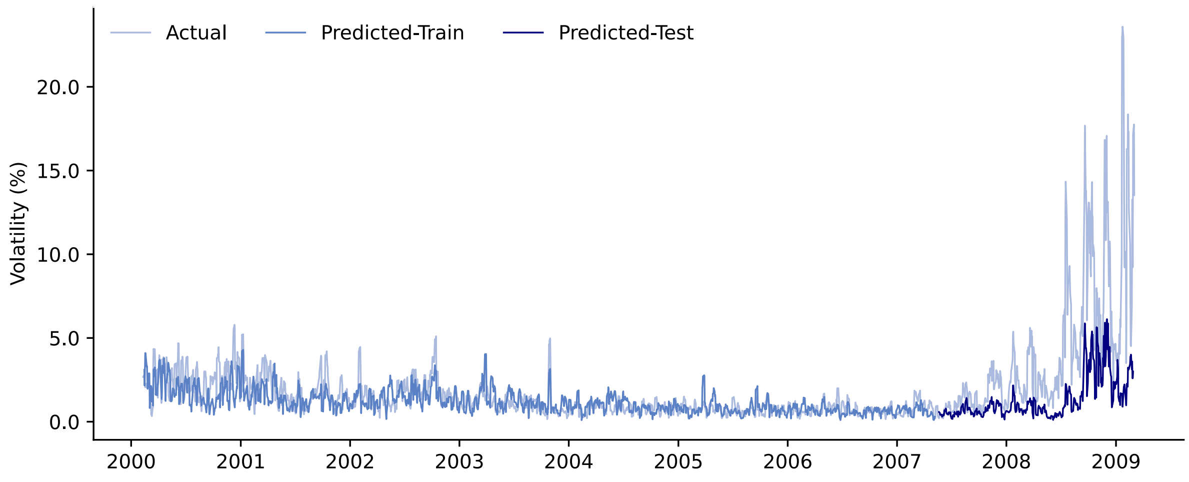


Fig 42: BAC price volatility prediction in bear market with OIL and WMT price feature inputs.

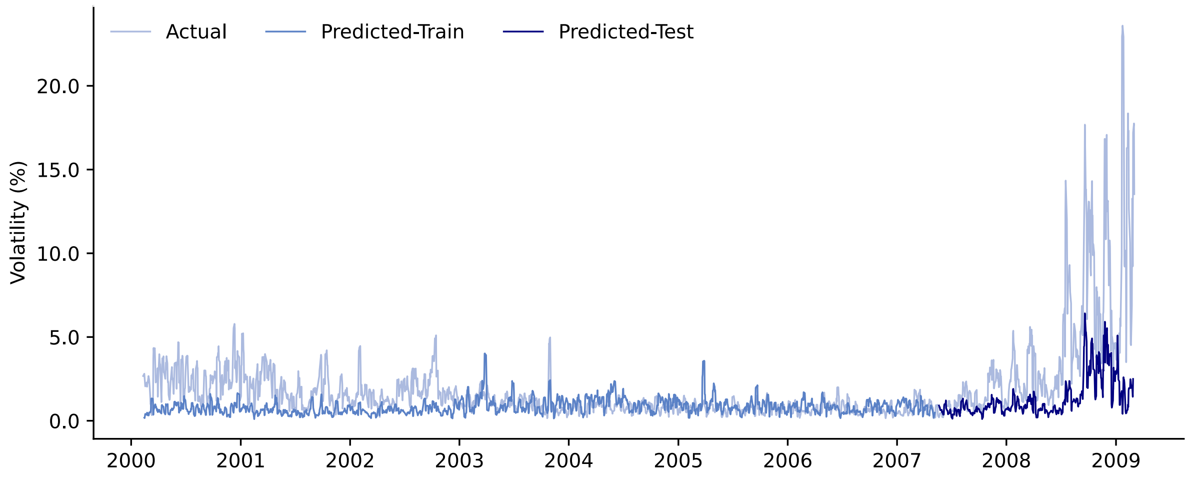


Fig 43: BAC price volatility prediction in bear market with OIL feature input.

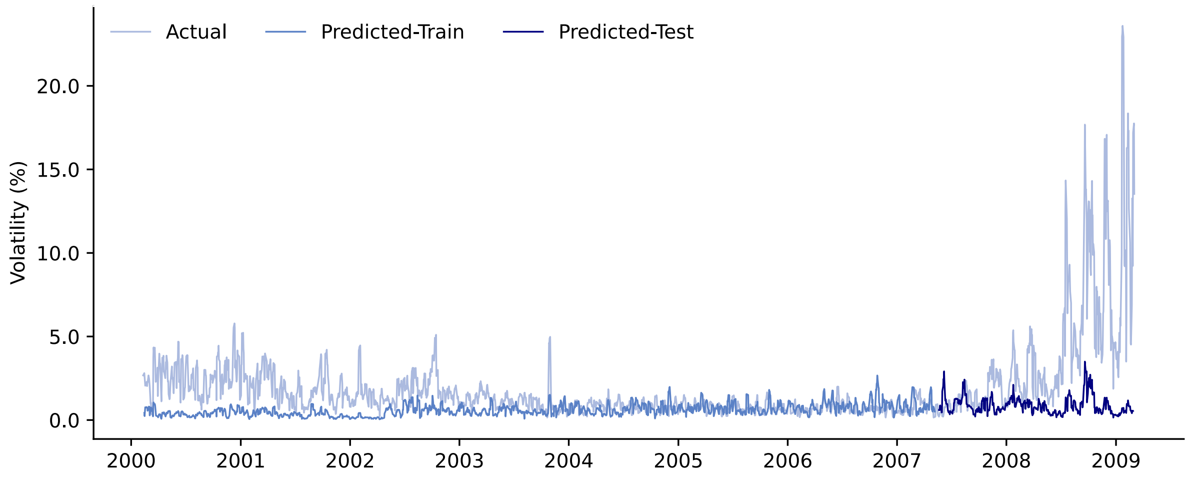


Fig 44: BAC price volatility prediction in bear market with WMT price feature input.

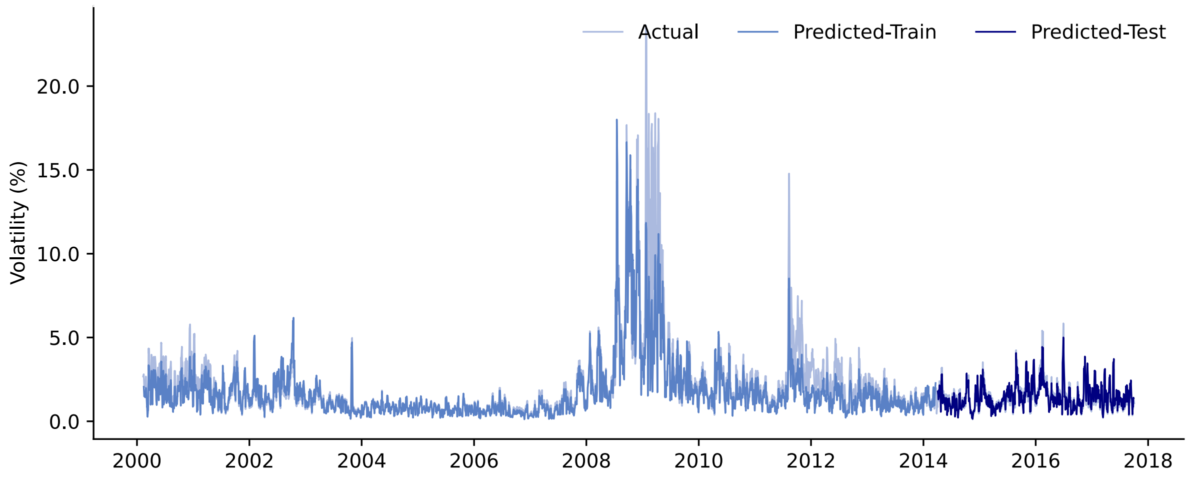


Fig 45: BAC price volatility prediction in bull market with C price feature input.

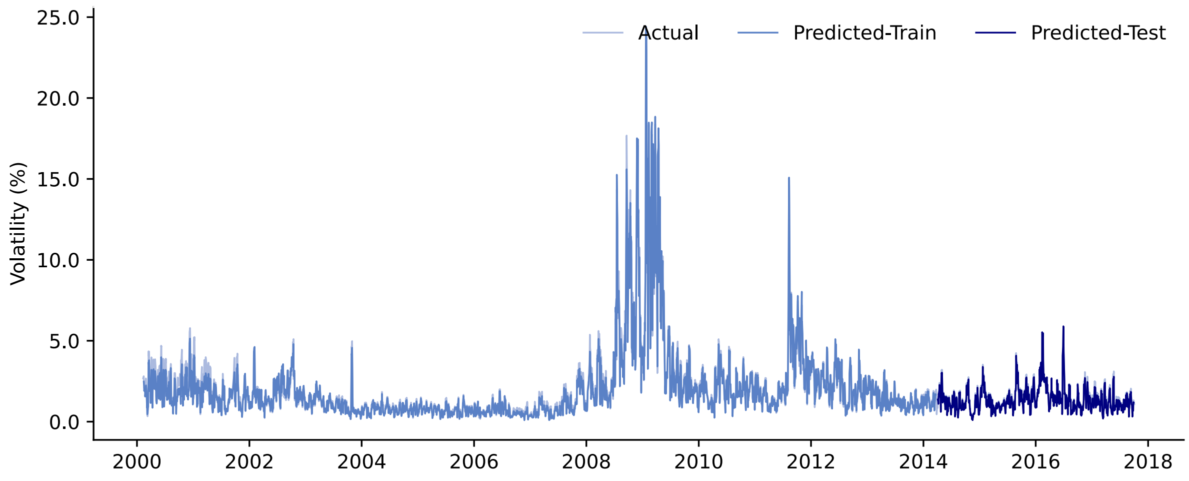


Fig 46: BAC price volatility prediction in bull market with DB and C price feature inputs.

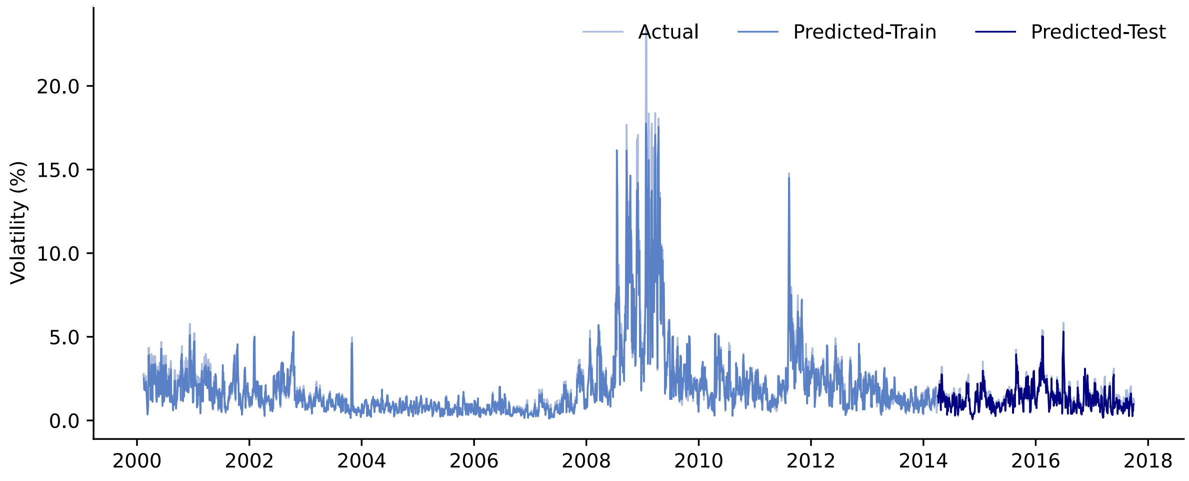


Fig 47: BAC price volatility prediction in bull market with DB, GS, and C and price feature inputs.

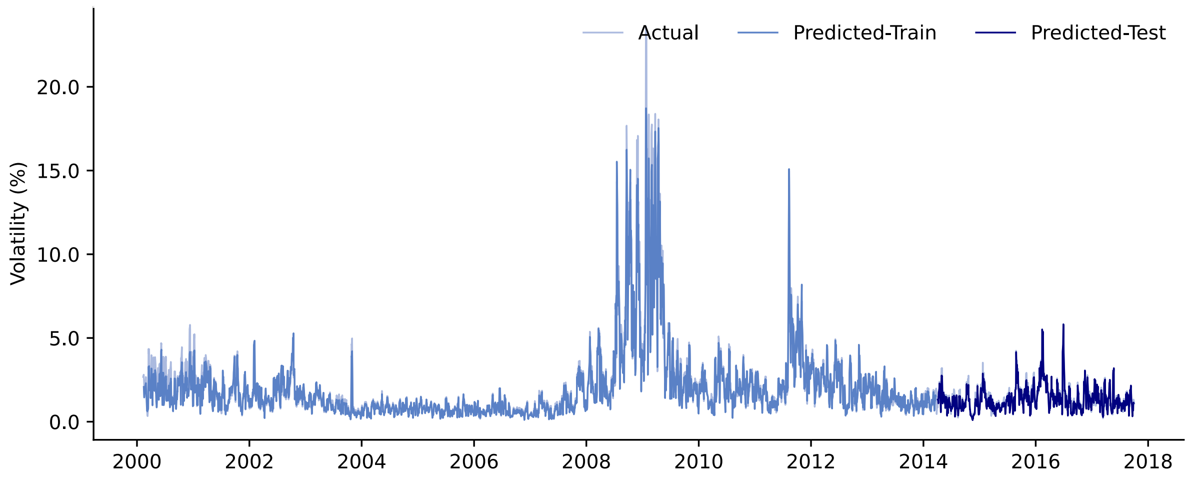


Fig 48: BAC price volatility prediction in bull market with DB, GS, C, and MS price feature inputs.

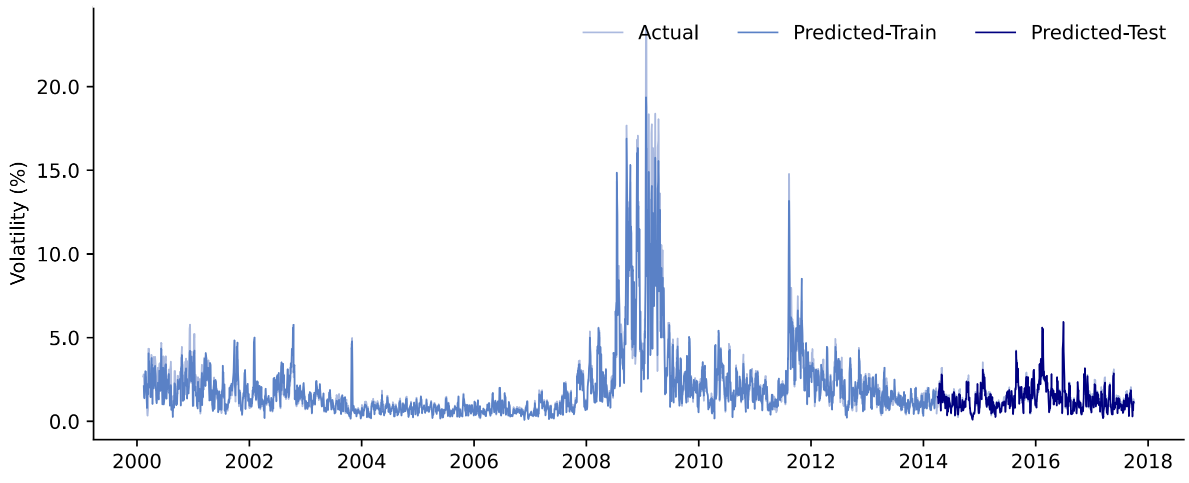


Fig 49: BAC price volatility prediction in bull market with DB, GS price feature inputs.

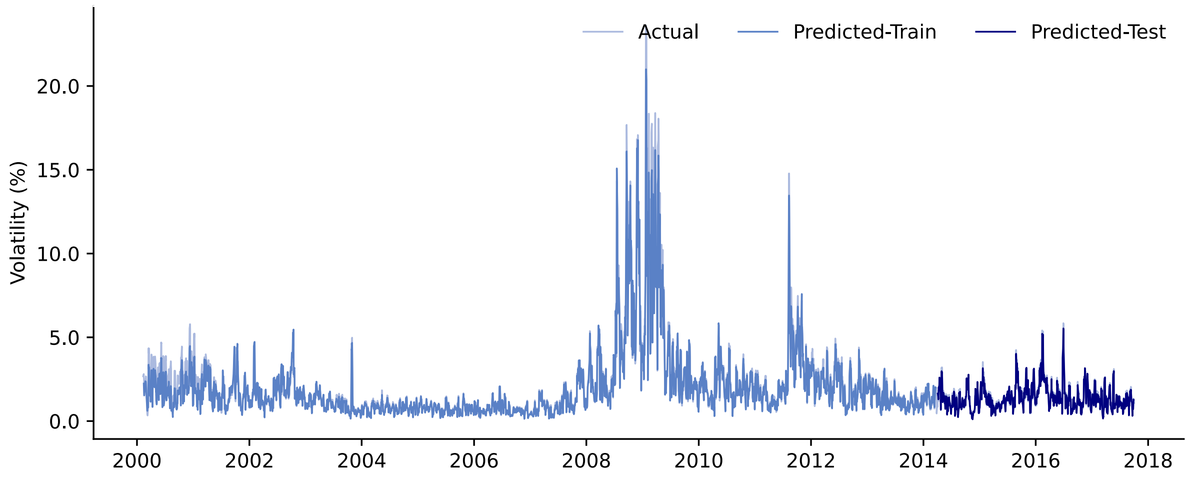


Fig 50: BAC price volatility prediction in bull market with DB and MS price feature inputs.

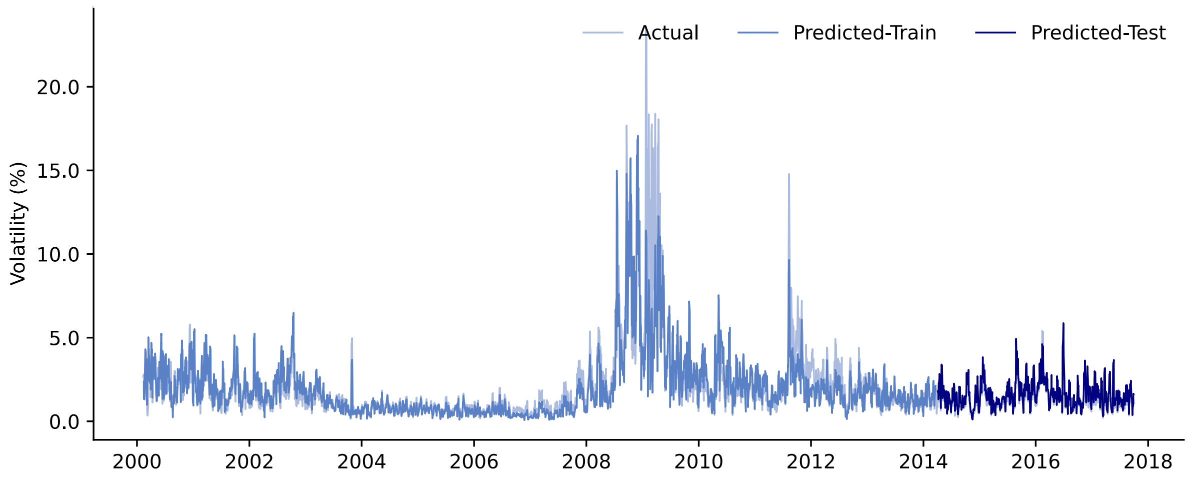


Fig 51: BAC price volatility prediction in bull market with DB price feature input

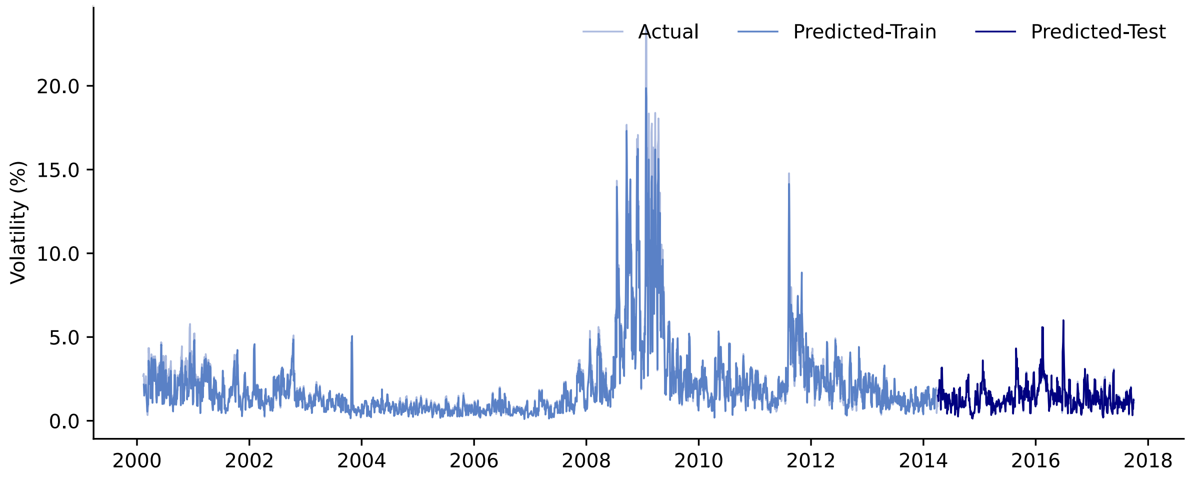


Fig 52: BAC price volatility prediction in bull market with GOLD and DB price feature inputs.

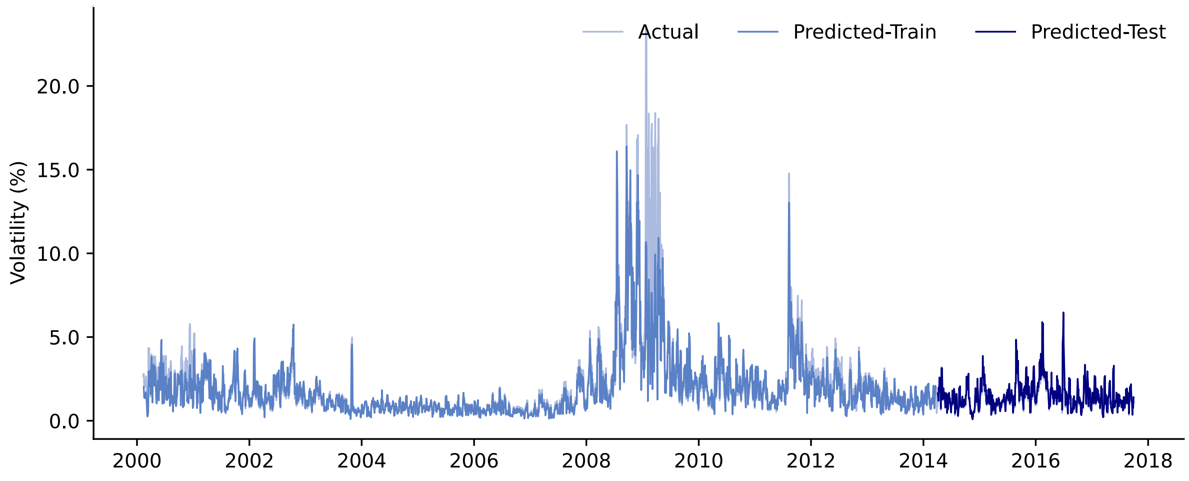


Fig 53: BAC price volatility prediction in bull market with GOLD and IBM price feature inputs.

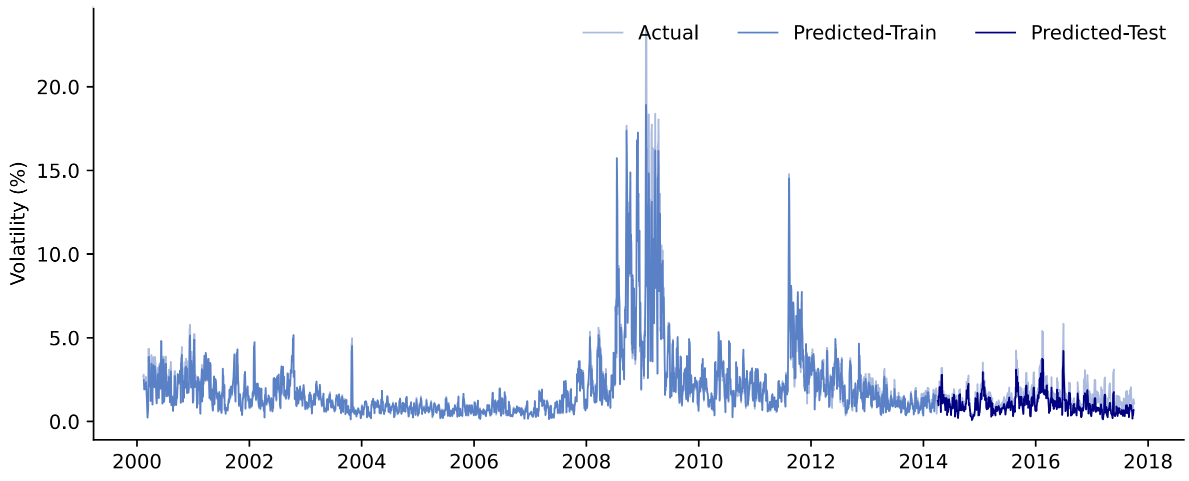


Fig 54: BAC price volatility prediction in bull market with GOLD and JPM price feature inputs.

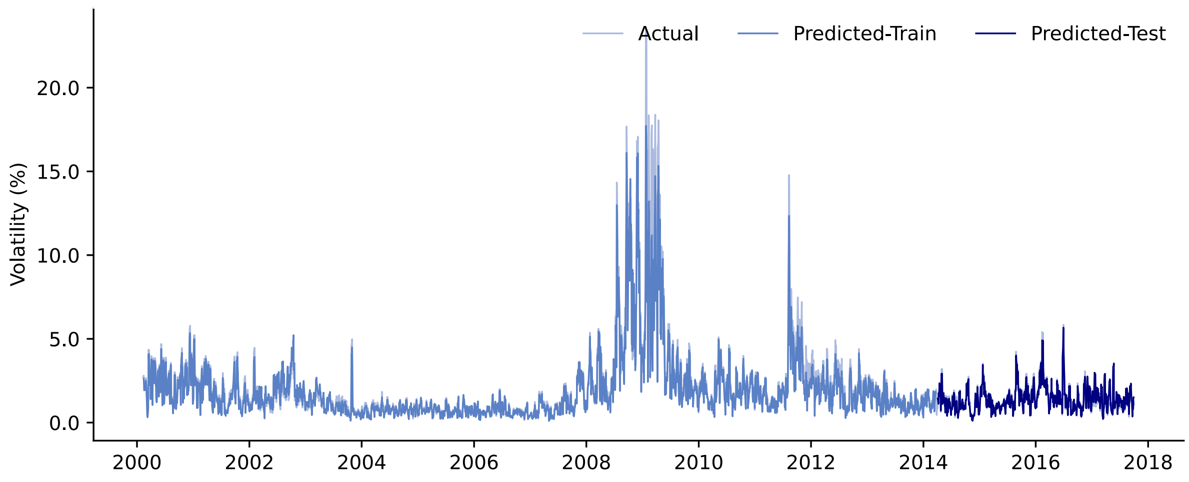


Fig 55: BAC price volatility prediction in bull market with GOLD and OIL price feature inputs.

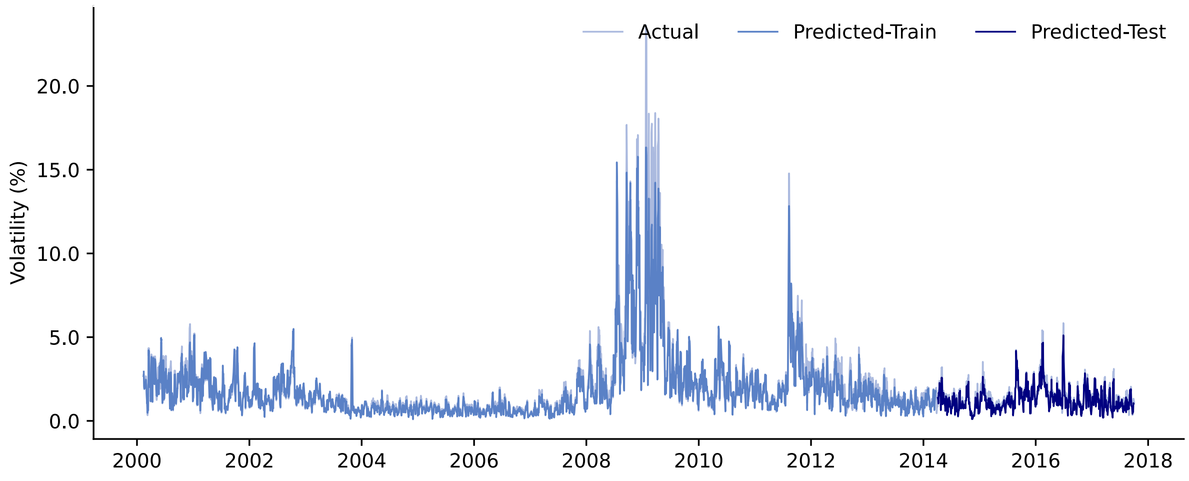


Fig 56: BAC price volatility prediction in bull market with GOLD and WMT price feature inputs.

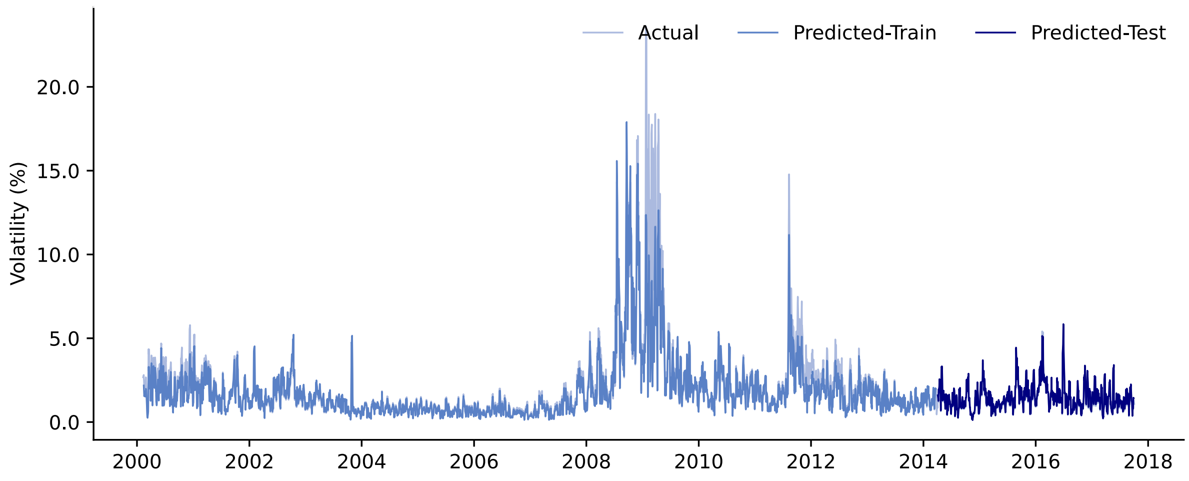


Fig 57: BAC price volatility prediction in bull market with GOLD price feature input.

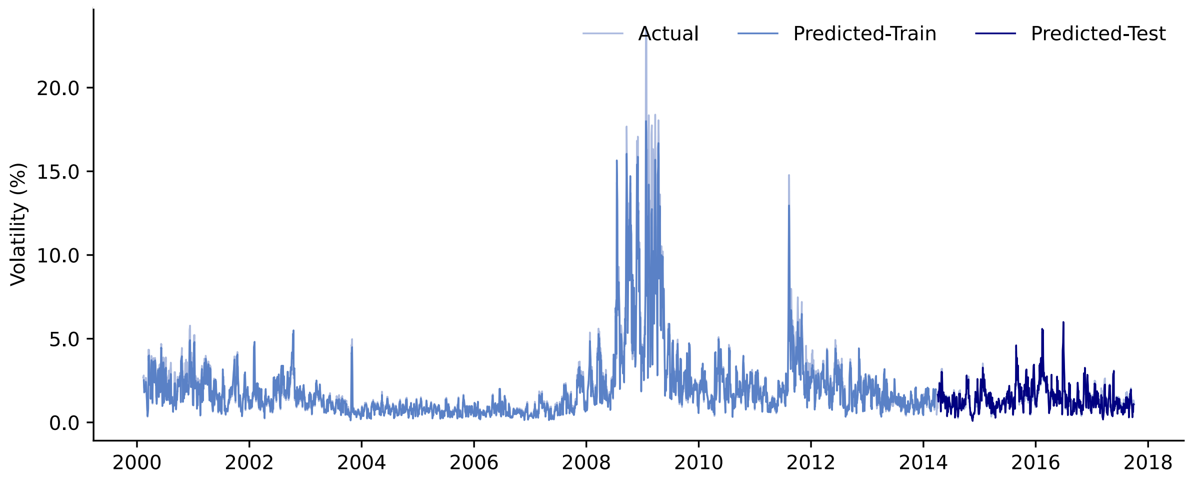


Fig 58: BAC price volatility prediction in bull market with GS and C price feature inputs.

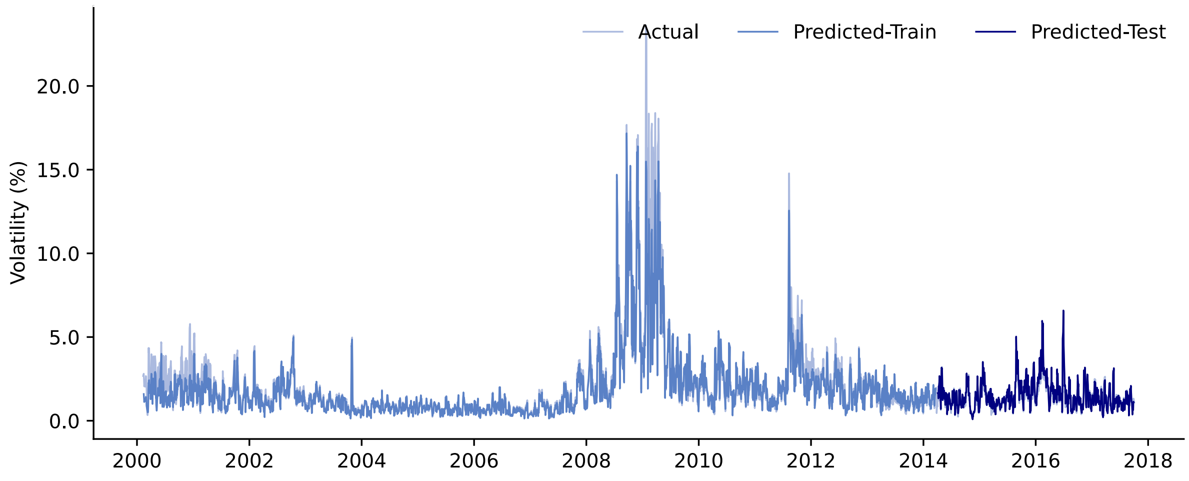


Fig 59: BAC price volatility prediction in bull market with GS, MS, and C price feature inputs.

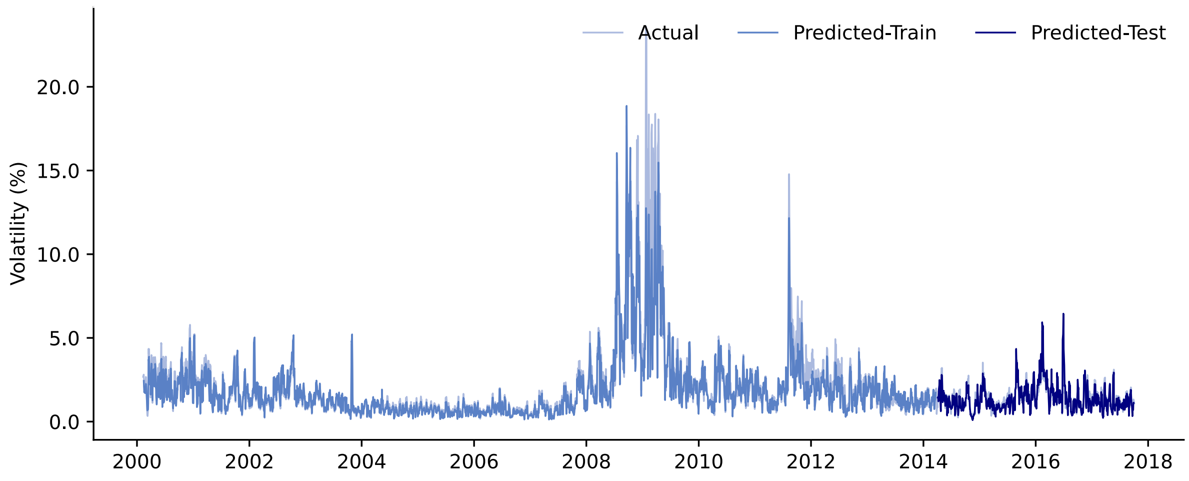


Fig 60: BAC price volatility prediction in bull market with GS and MS price feature inputs.

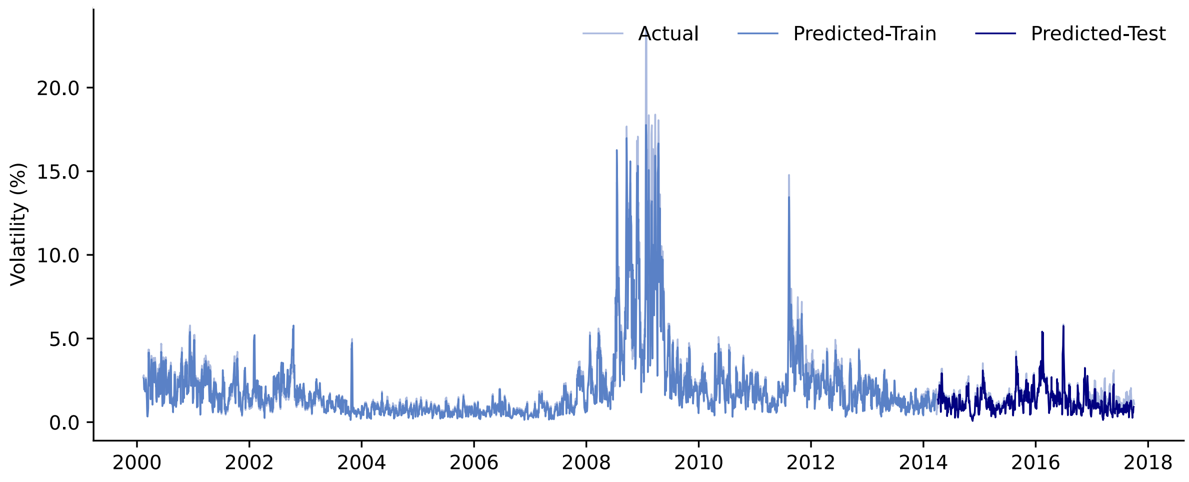


Fig 61: BAC price volatility prediction in bull market with GS price feature inputs.

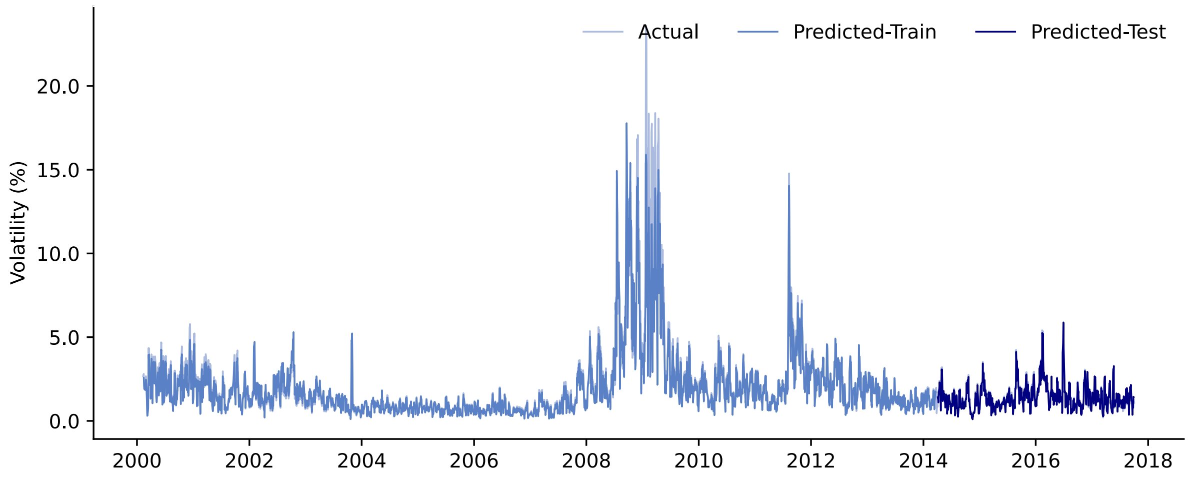


Fig 62: BAC price volatility prediction in bull market with IBM price feature input.

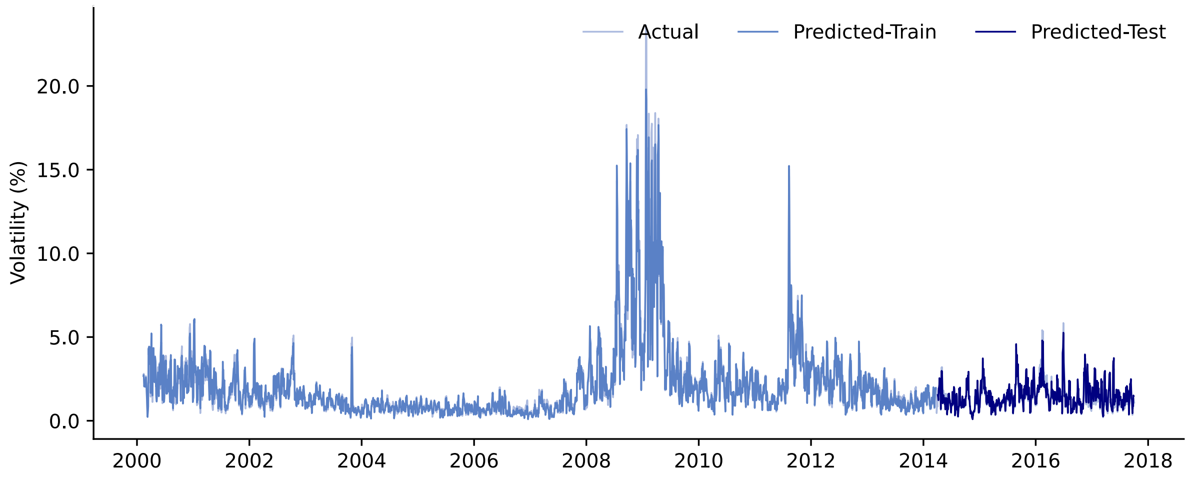


Fig 63: BAC price volatility prediction in bull market with JPM and C price feature inputs.

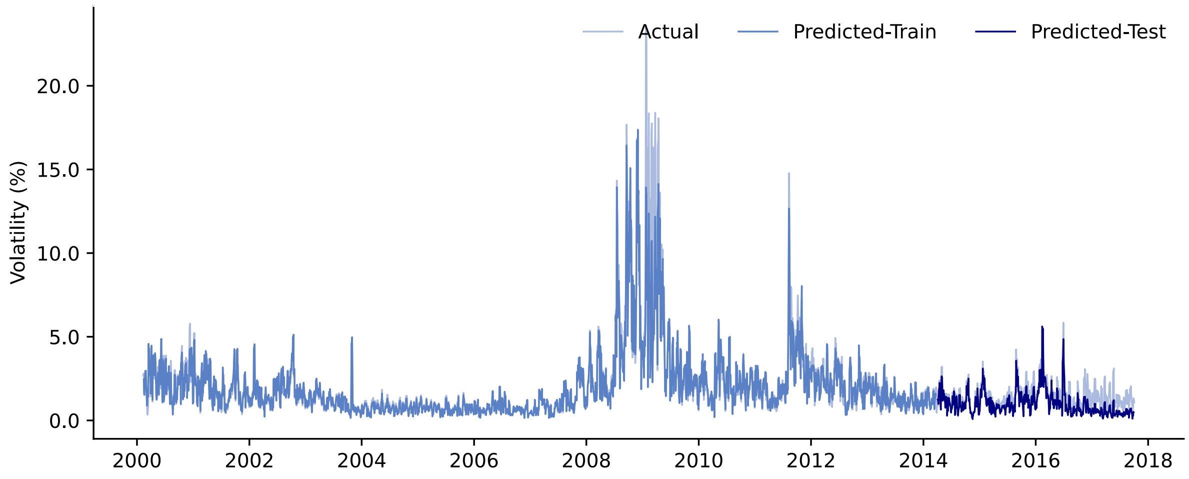


Fig 64: BAC price volatility prediction in bull market with JPM and DB price feature inputs.

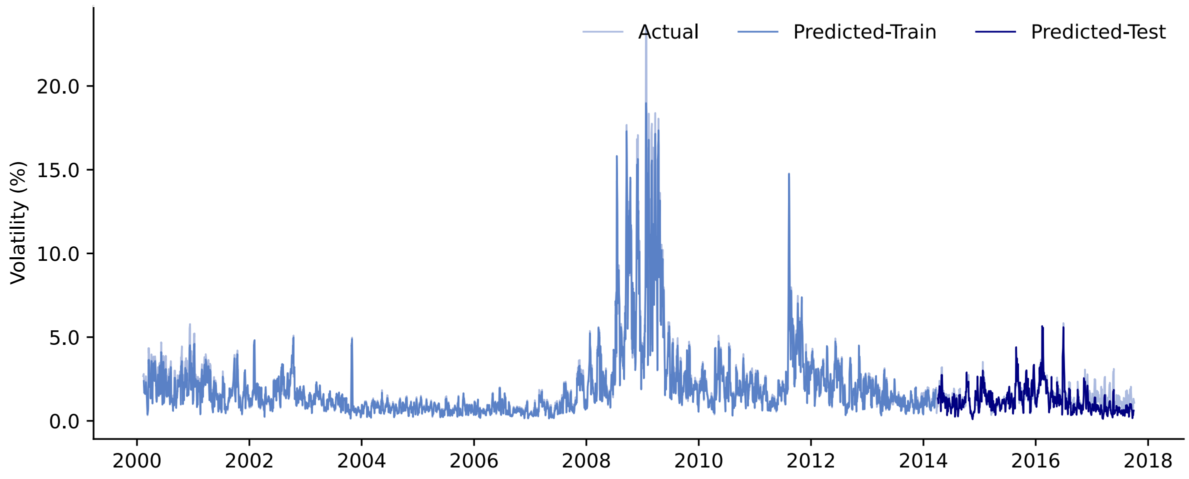


Fig 65: BAC price volatility prediction in bull market with JPM, GS and C price feature inputs.

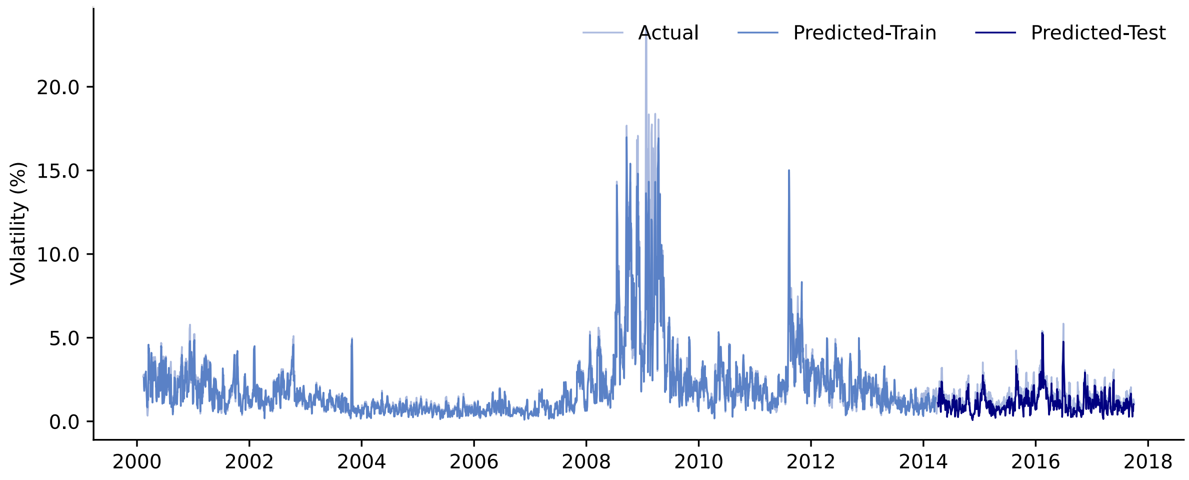


Fig 66: BAC price volatility prediction in bull market with JPM, GS and DB price feature inputs.

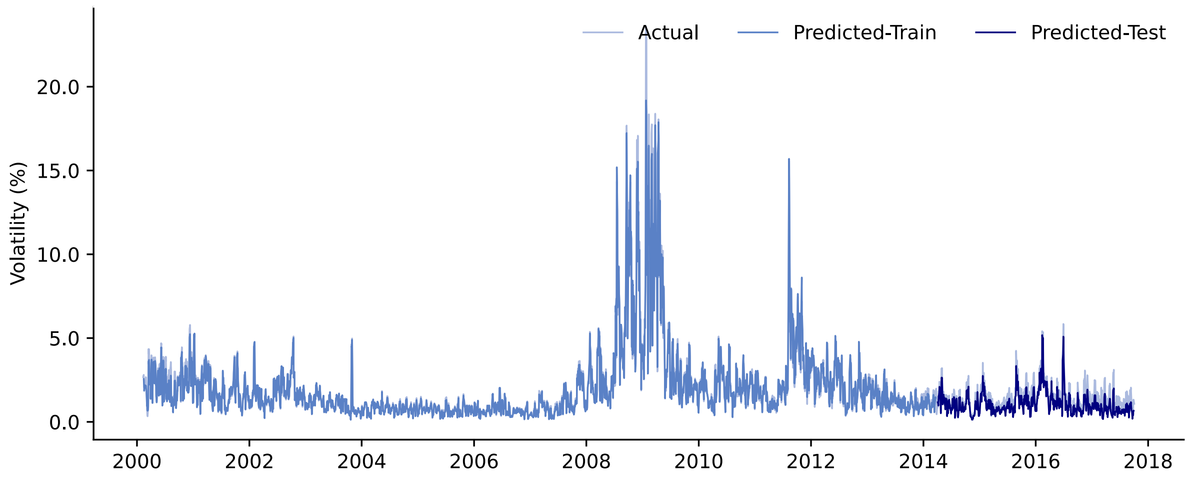


Fig 67: BAC price volatility prediction in bull market with JPM, GS, and MS price feature inputs.

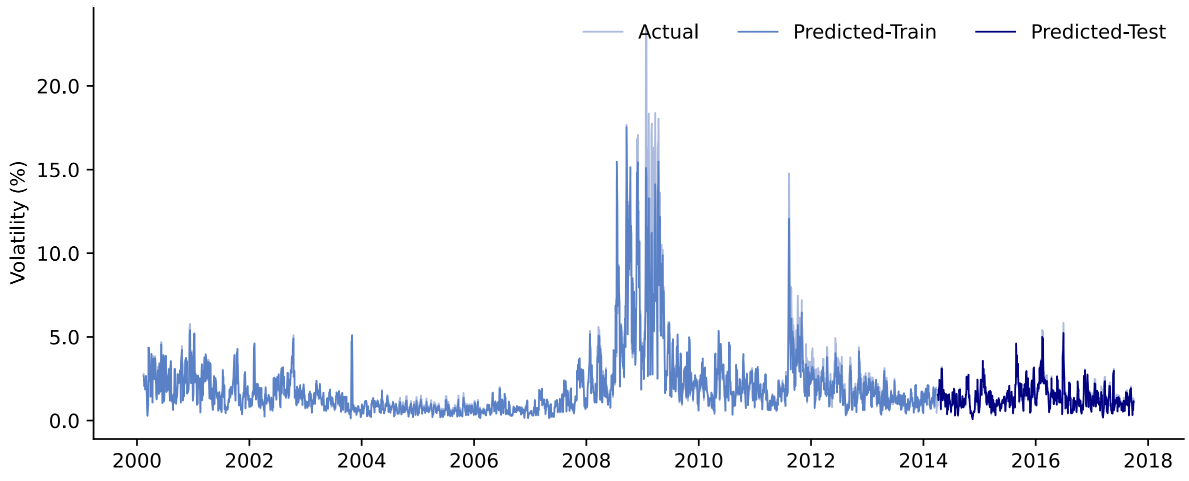


Fig 68: BAC price volatility prediction in bull market with JPM and GS price feature inputs.

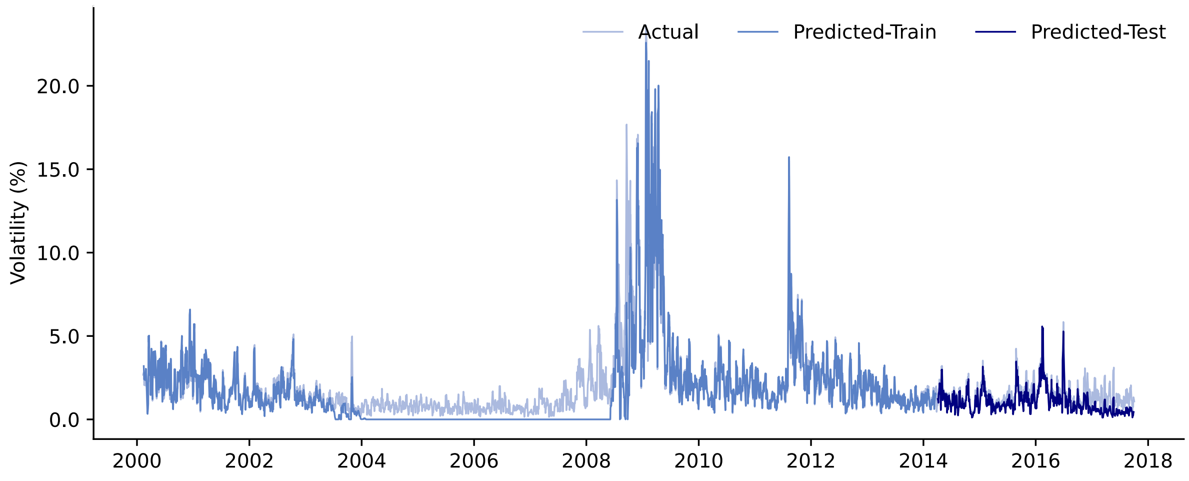


Fig 69: BAC price volatility prediction in bull market with JPM and MS price feature inputs.

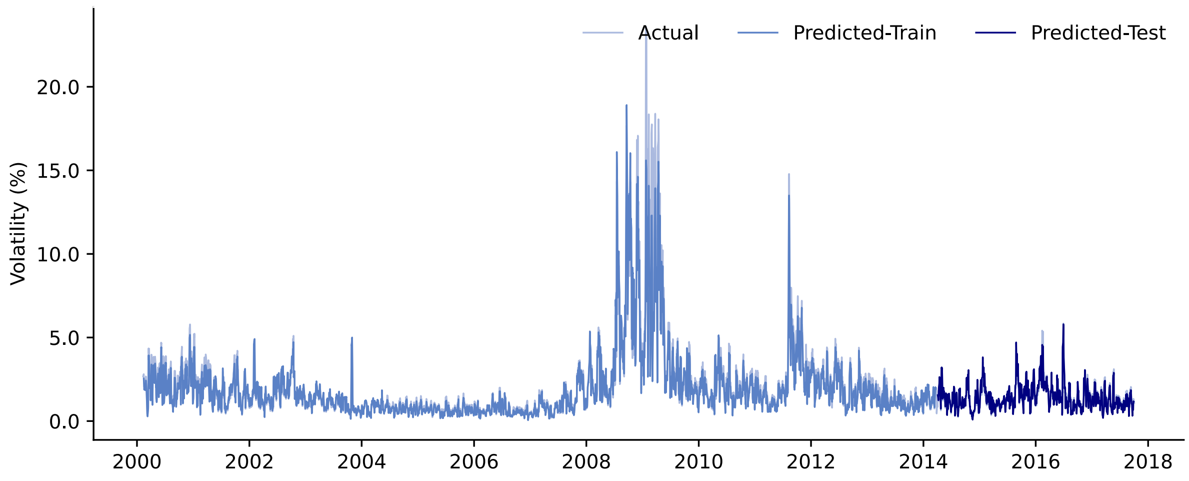


Fig 70: BAC price volatility prediction in bull market with JPM price feature input.

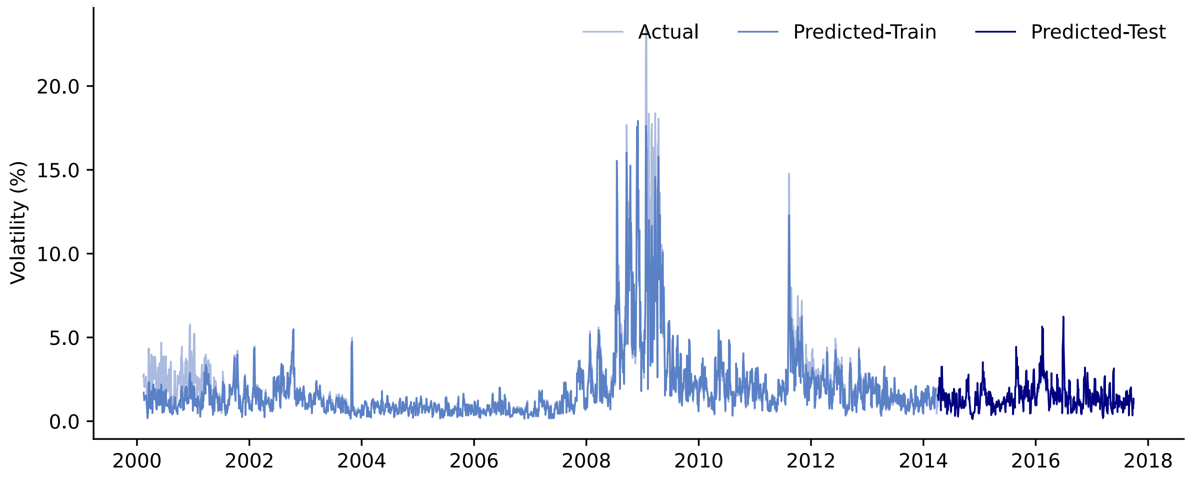


Fig 71: BAC price volatility prediction in bull market with MS and C price feature inputs.

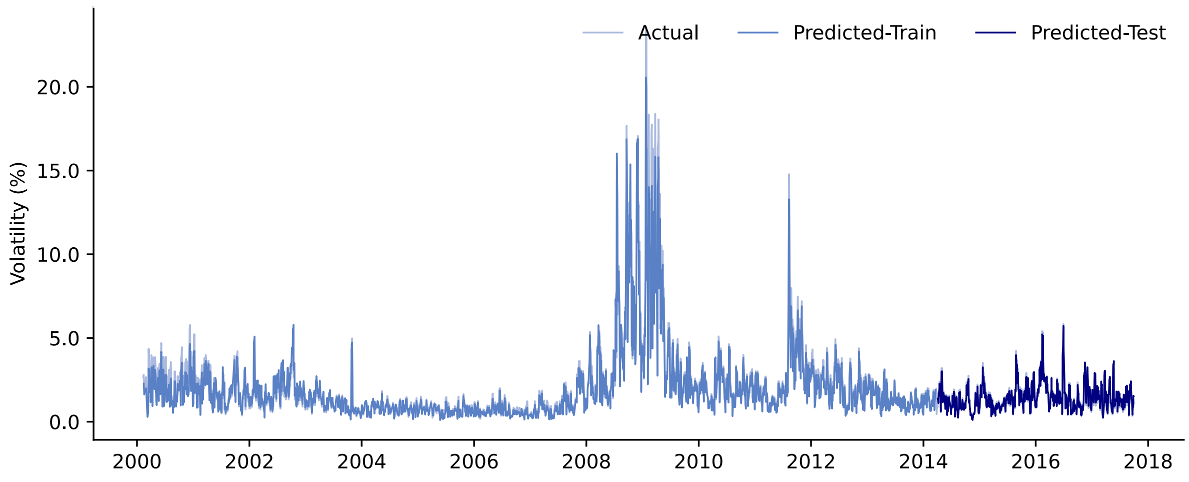


Fig 72: BAC price volatility prediction in bear market with MS price feature input.

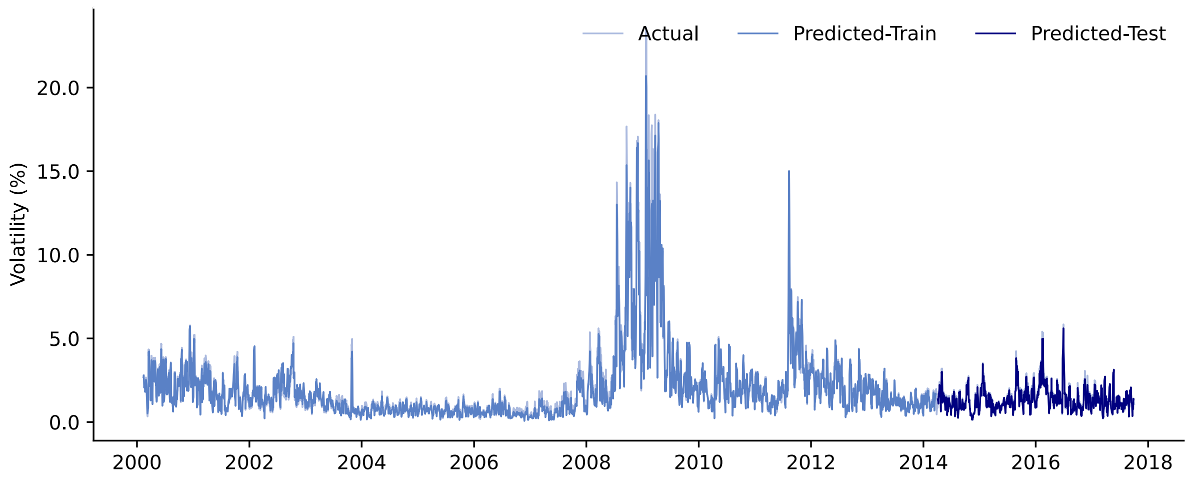


Fig 73: BAC price volatility prediction in bull market with OIL and DB price feature inputs.

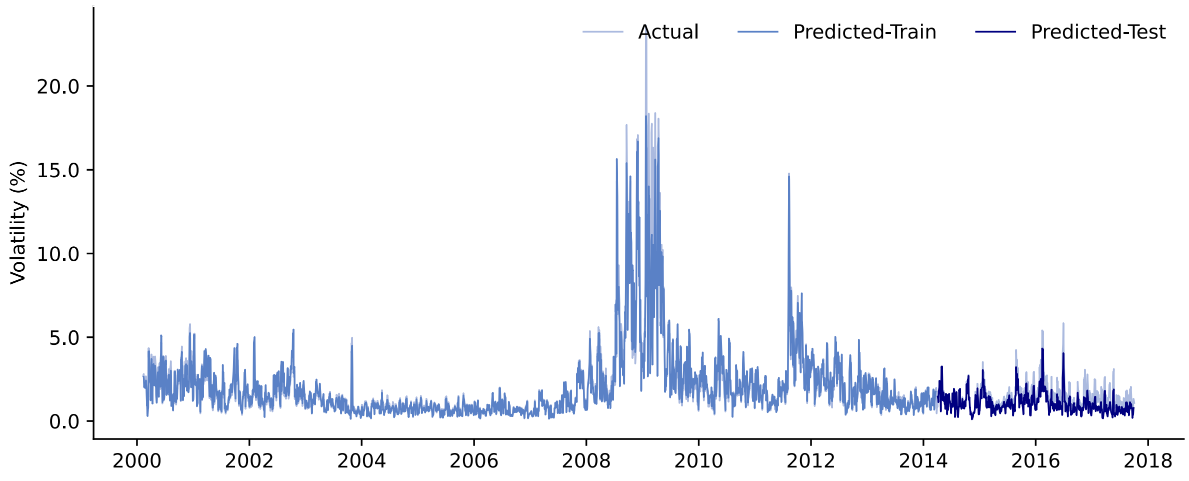


Fig 74: BAC price volatility prediction in bull market with OIL and IBM price feature inputs.

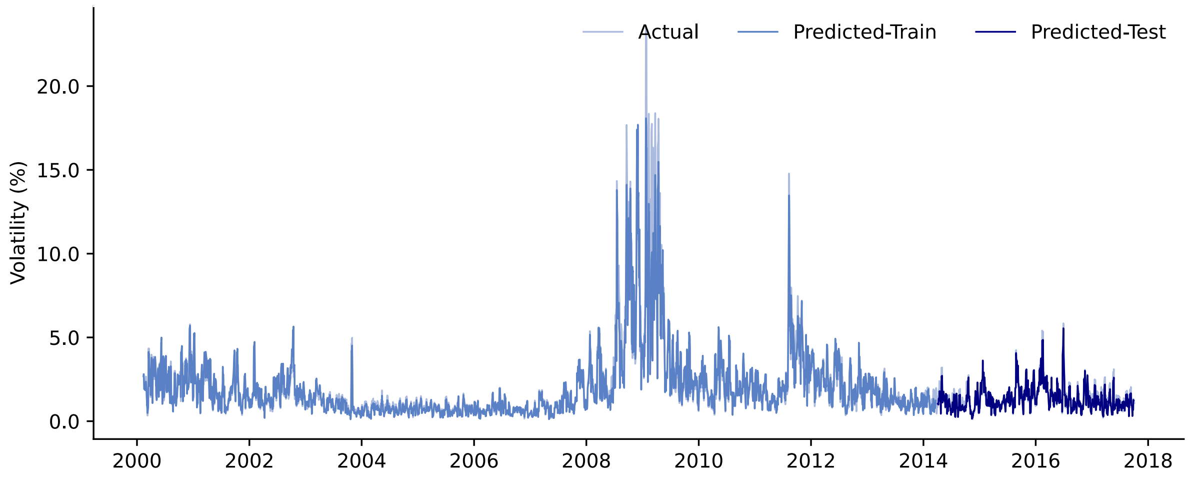


Fig 75: BAC price volatility prediction in bull market with OIL and JPM price feature inputs.

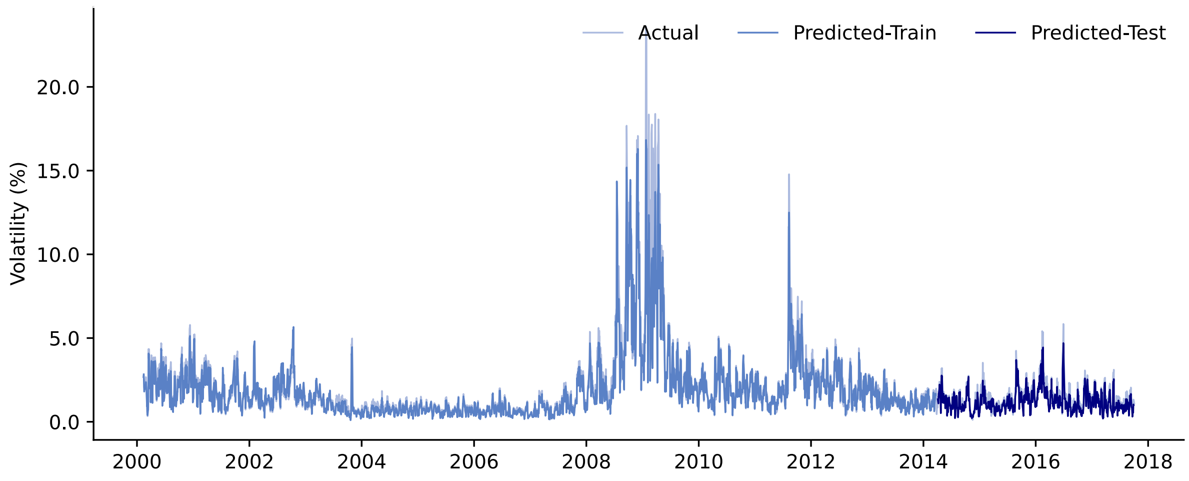


Fig 76: BAC price volatility prediction in bull market with OIL and WMT price feature inputs.

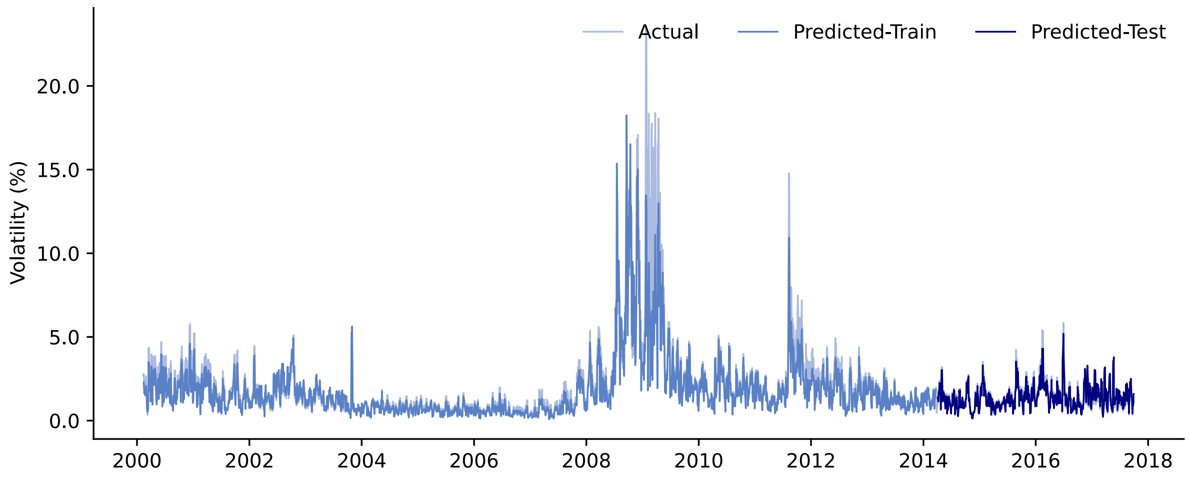


Fig 77: BAC price volatility prediction in bull market with OIL feature input.

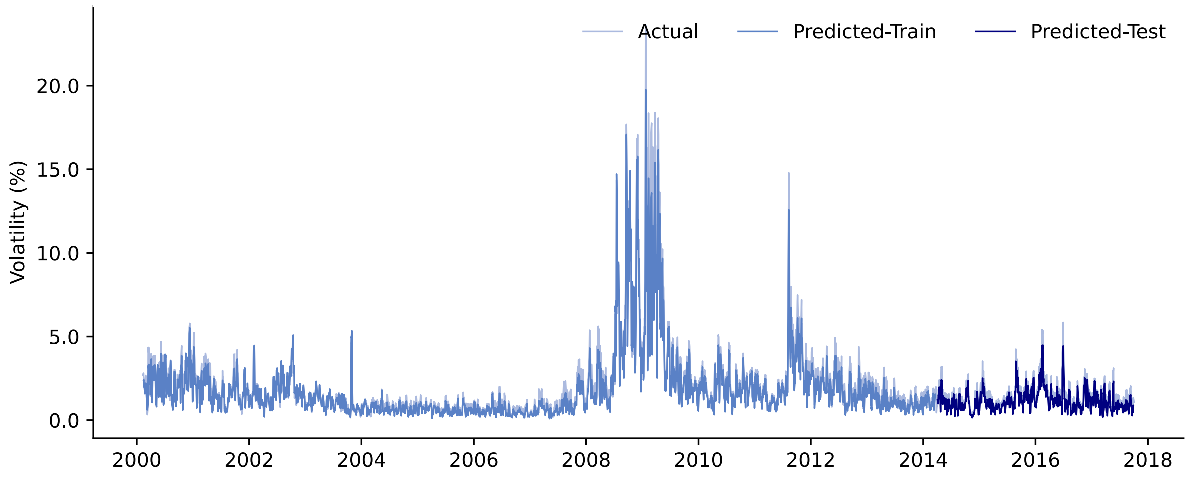


Fig 78: BAC price volatility prediction in bull market with WMT price feature input.

A.2.2 Multi-Task Deep Learning – Chapter 6

A.2.2.1 Benchmark testing

A.2.2.1.1 GARCH

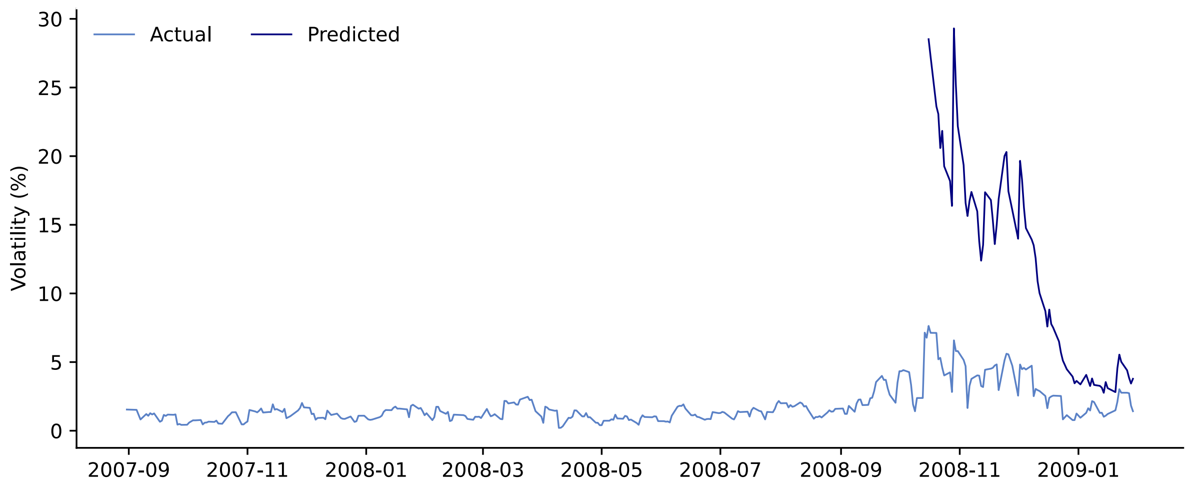


Fig 79: DJI volatility prediction in bear market.

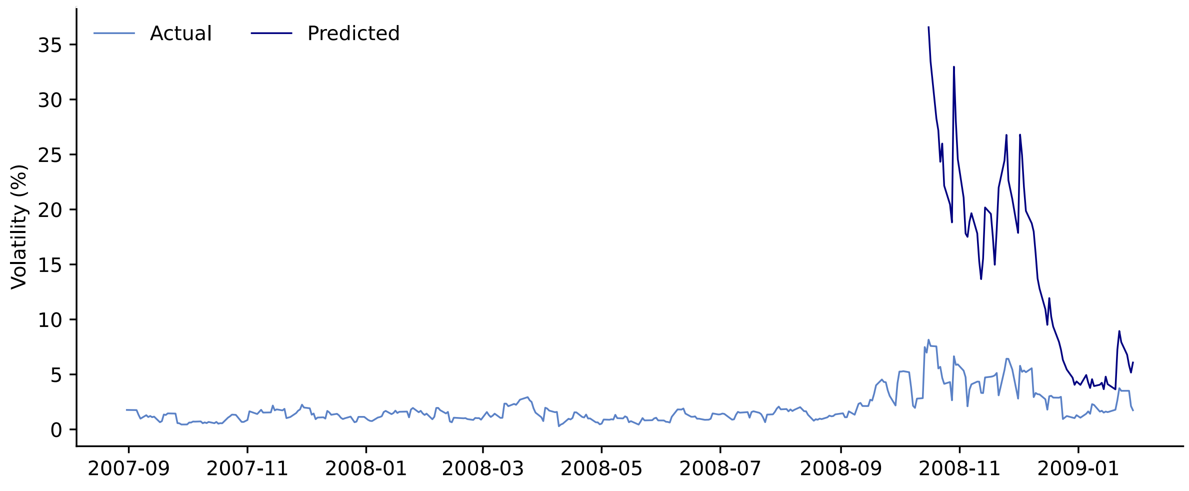


Fig 80: GSPC volatility prediction in bear market.

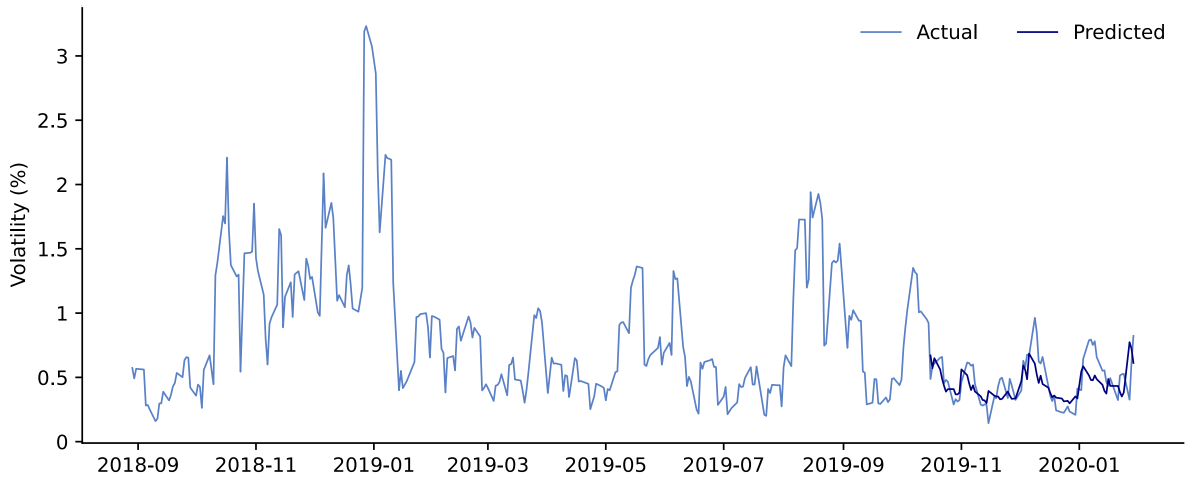


Fig 81: DJI volatility prediction in bull market.

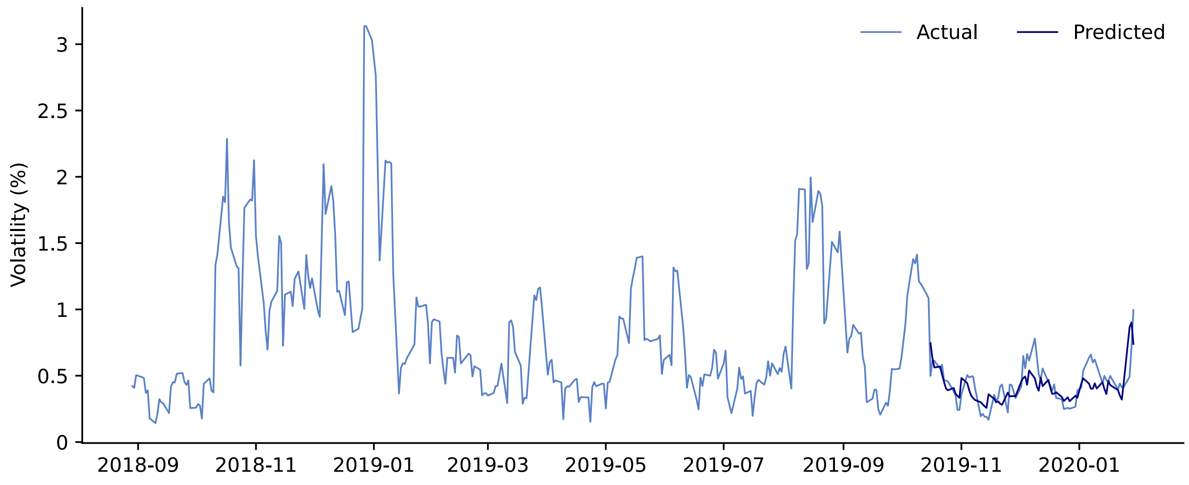


Fig 82: GSPC volatility prediction in bull market.

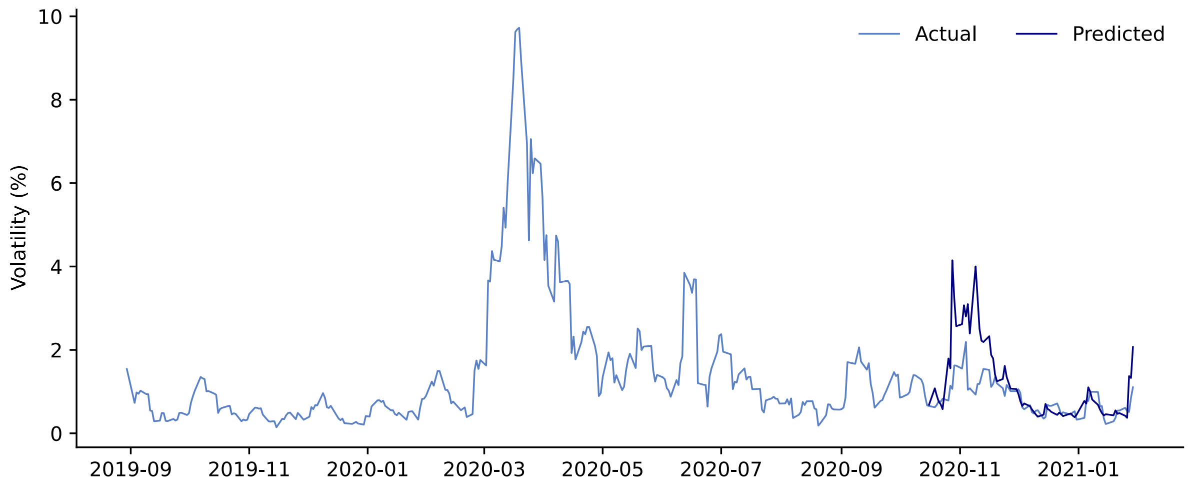


Fig 83: DJI volatility prediction in volatile market.

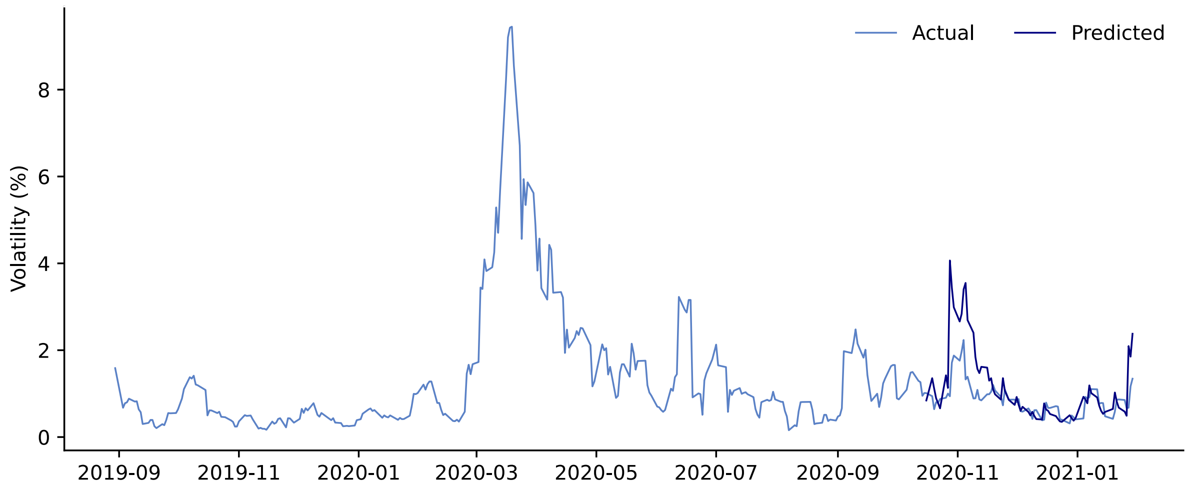


Fig 84: GSPC volatility prediction in volatile market.

A.2.2.1.2 ARIMA

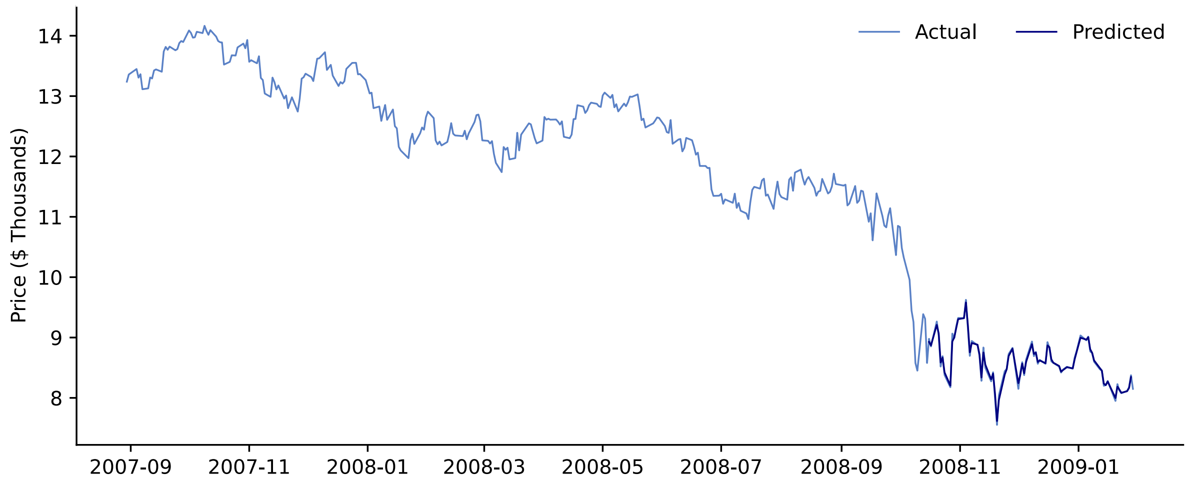


Fig 85: DJI volatility prediction in bear market.

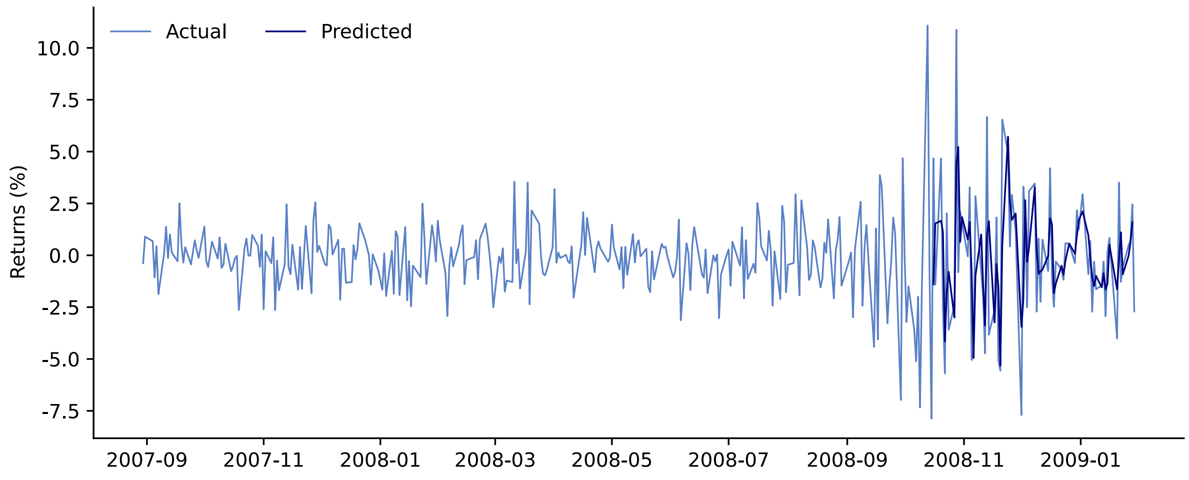


Fig 86: DJI returns prediction in bear market.

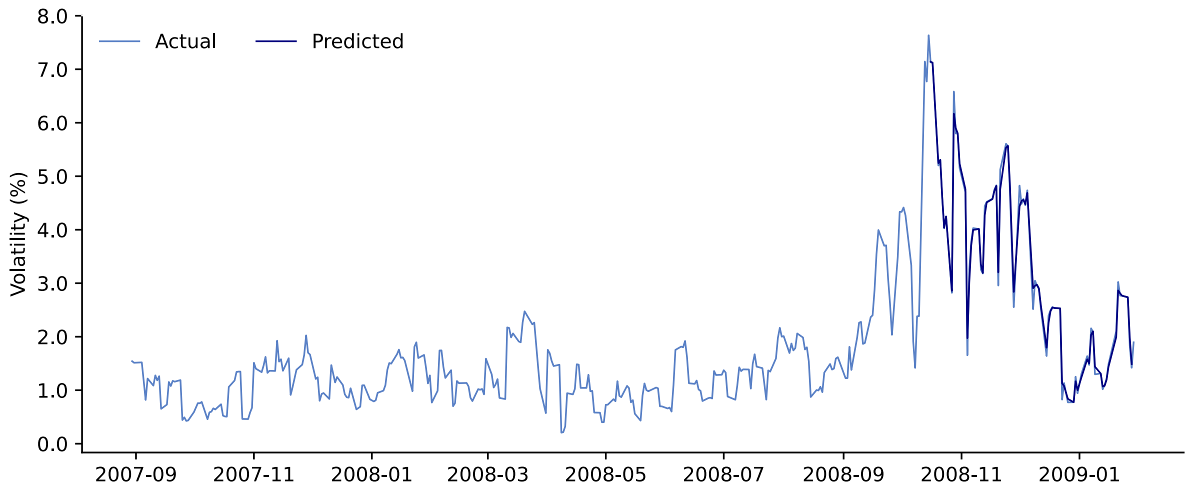


Fig 87: DJI volatility prediction in bear market.

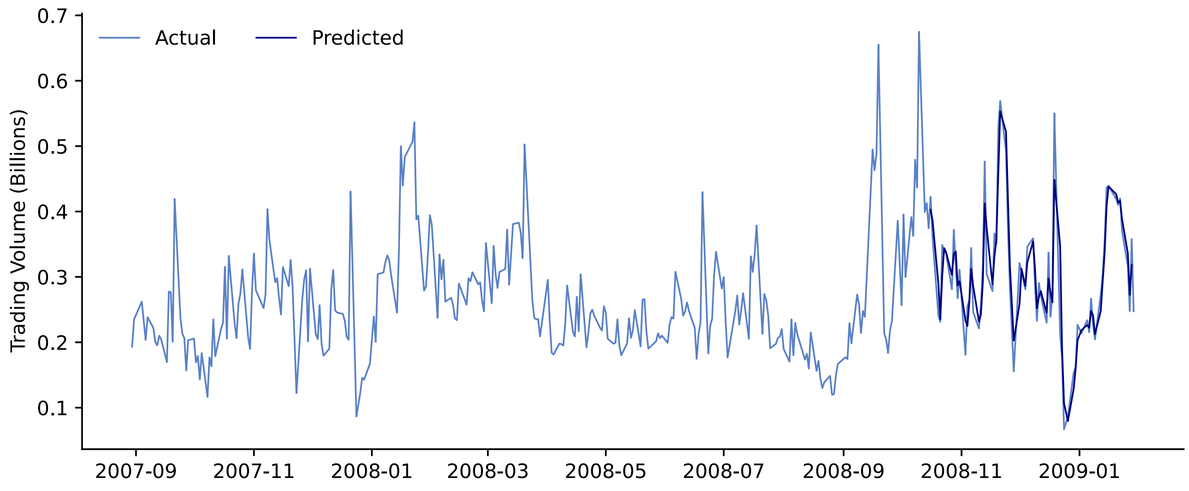


Fig 88: DJI trading volume prediction in bear market.

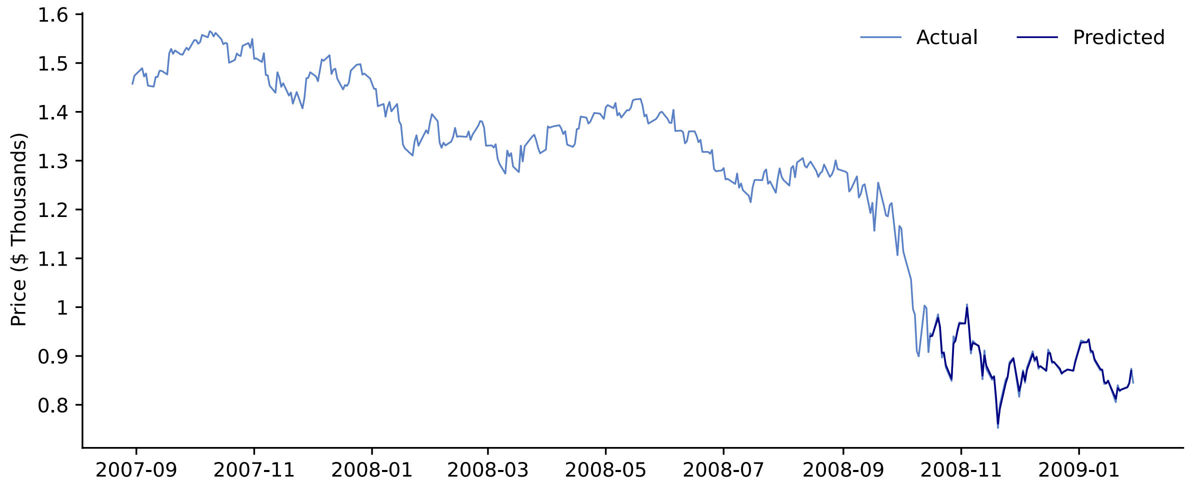


Fig 89: GSPC price prediction in bear market.

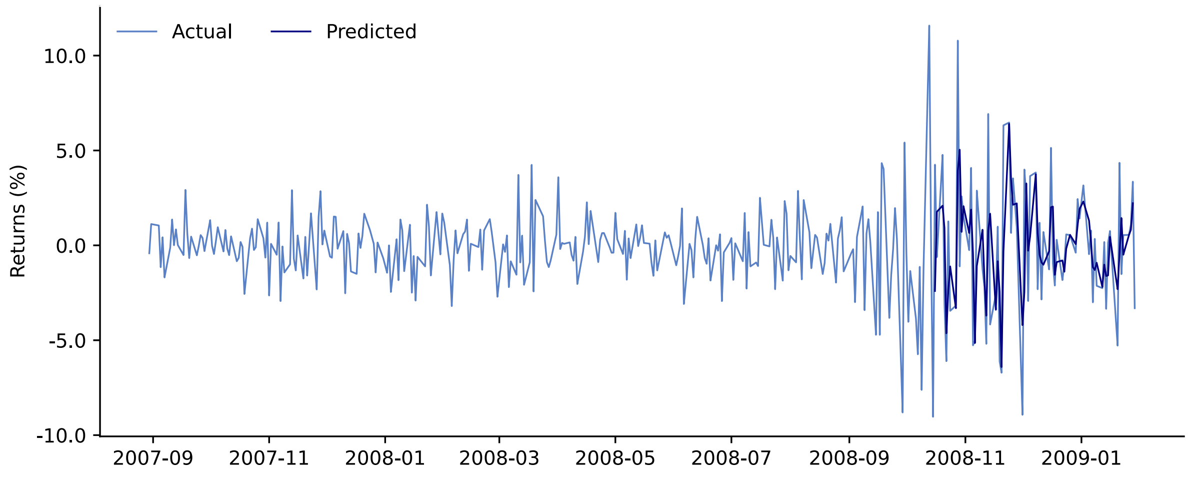


Fig 90: GSPC returns prediction in bear market.

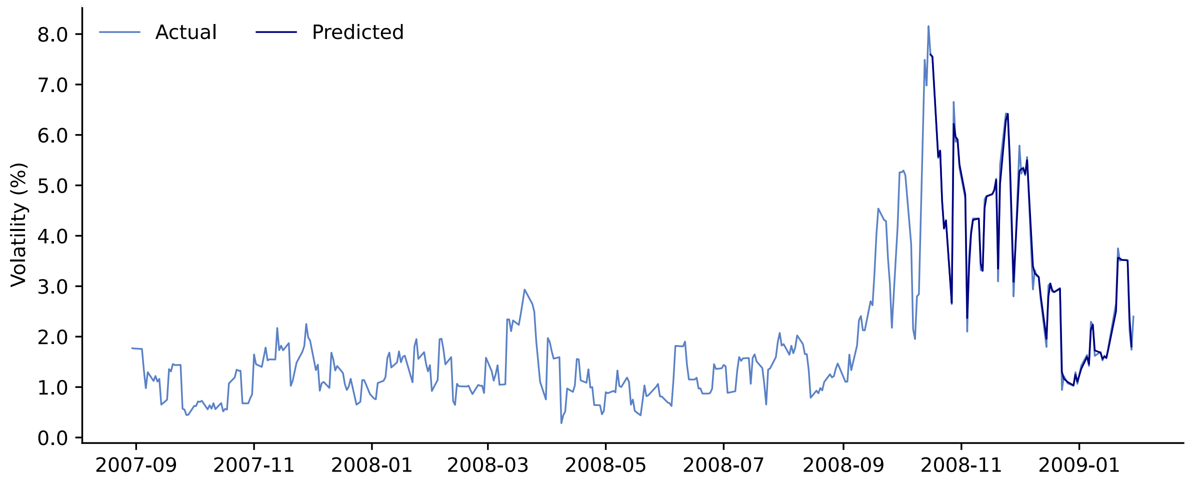


Fig 91: GSPC volatility prediction in bear market.

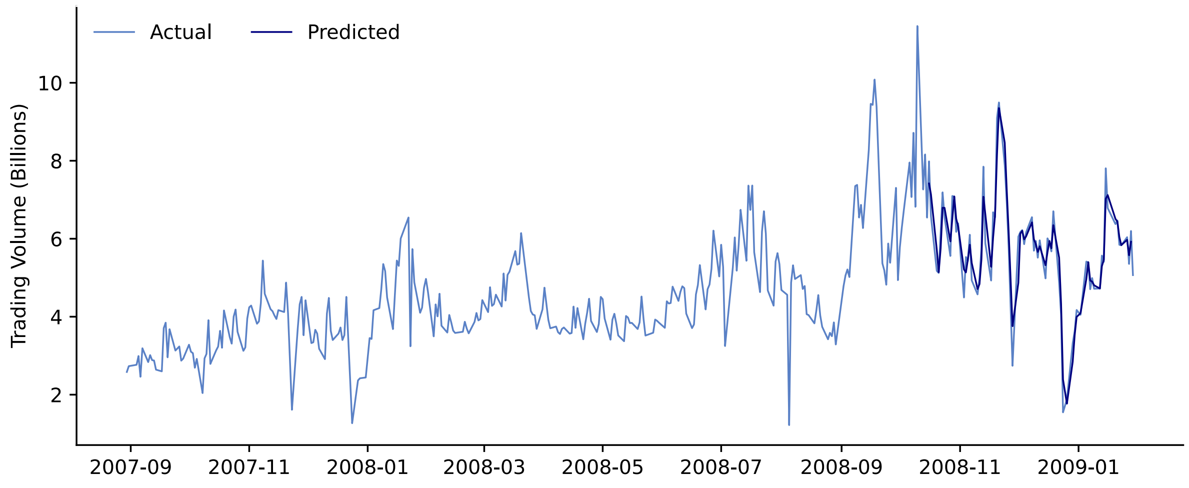


Fig 92: GSPC trading volume prediction in bear market.

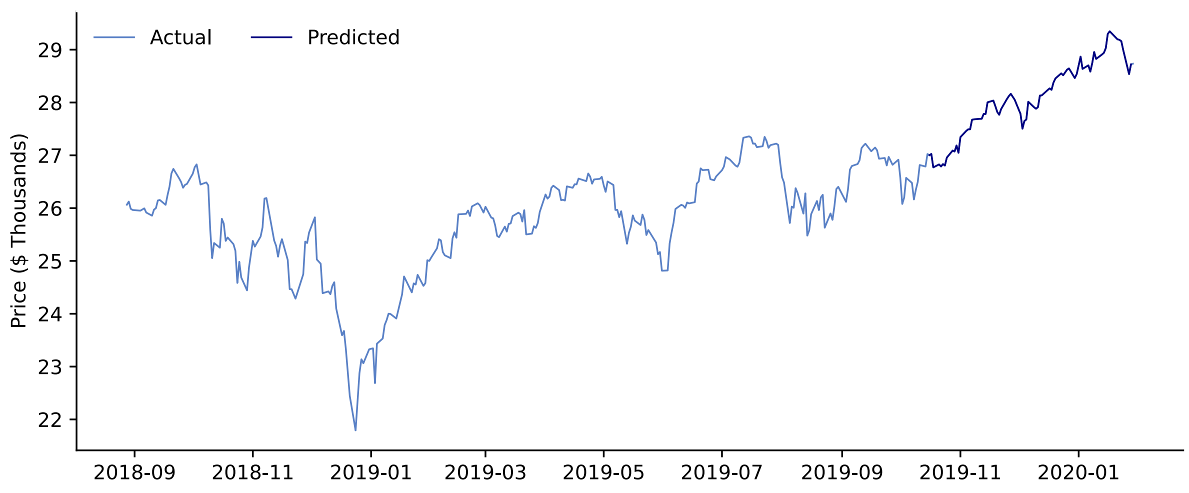


Fig 93: DJI price prediction in bull market.

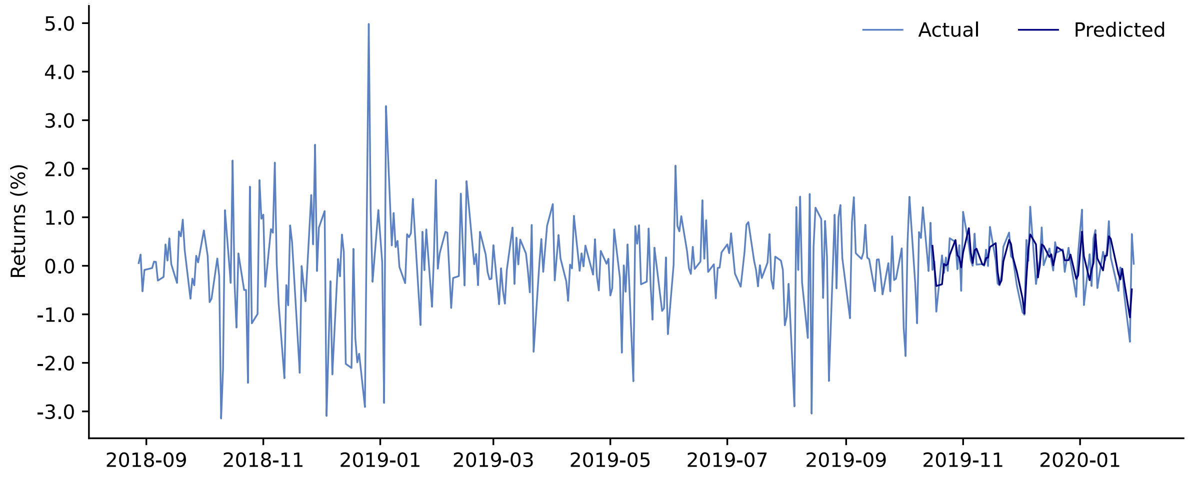


Fig 94: DJI returns prediction in bull market.

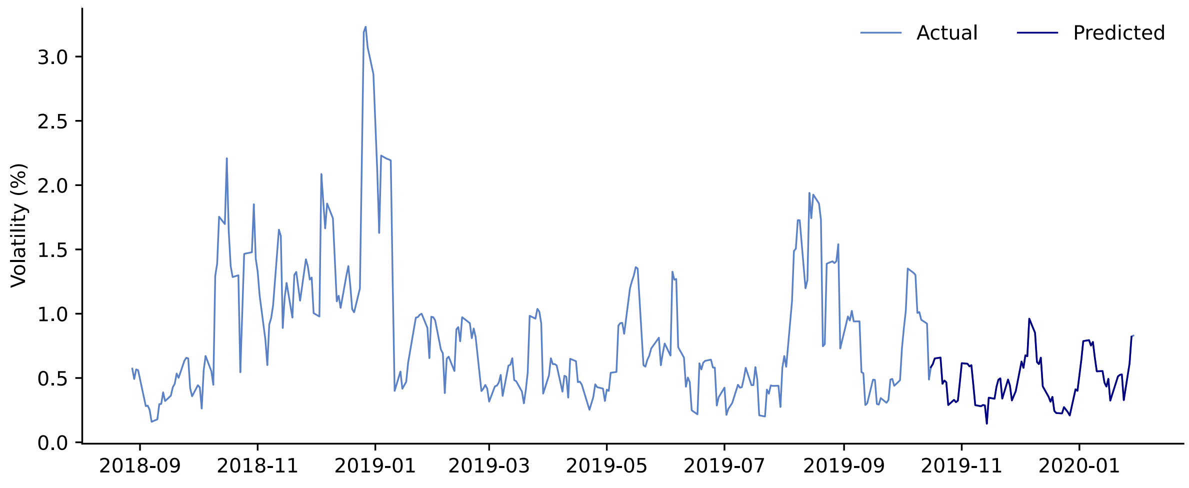


Fig 95: DJI volatility prediction in bull market.

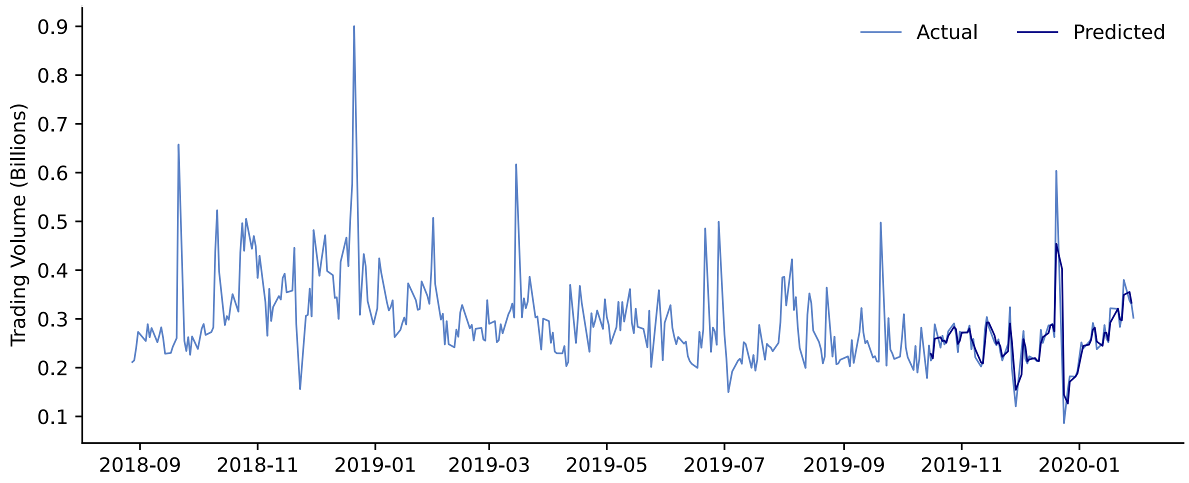


Fig 96: DJI trading volume prediction in bull market.

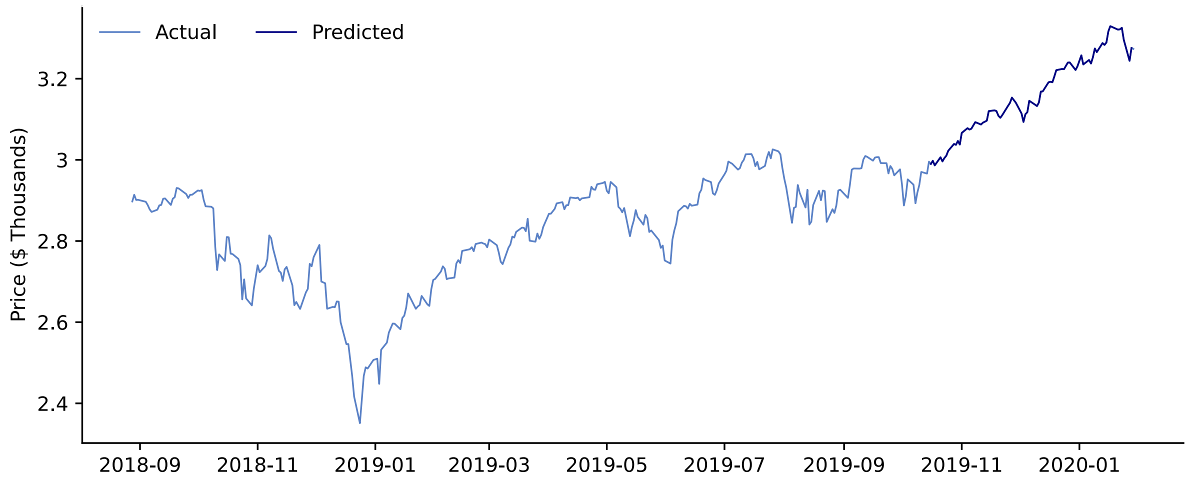


Fig 97: GSPC price prediction in bear market.

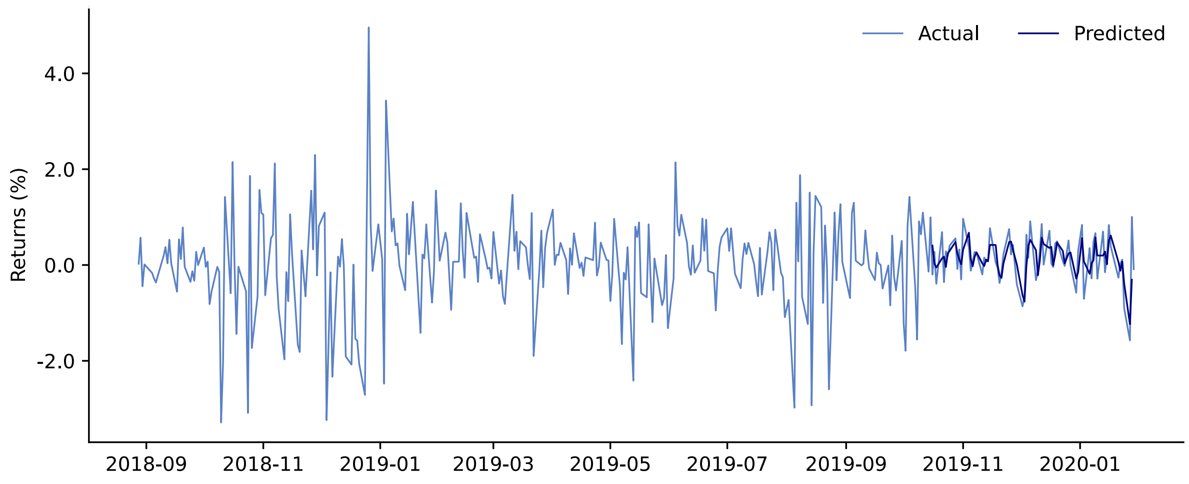


Fig 98: GSPC returns prediction in bear market.

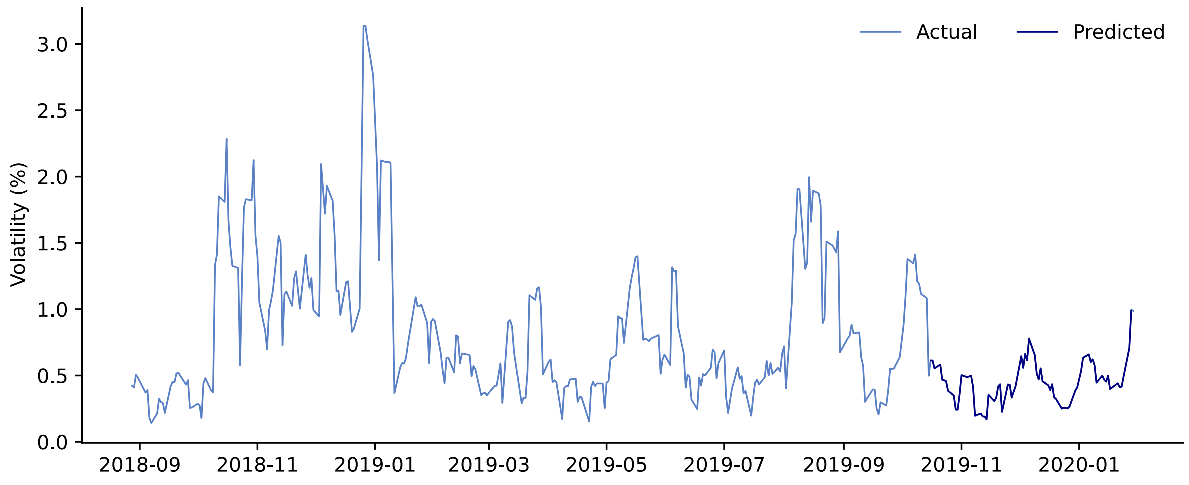


Fig 99: GSPC volatility prediction in bear market.

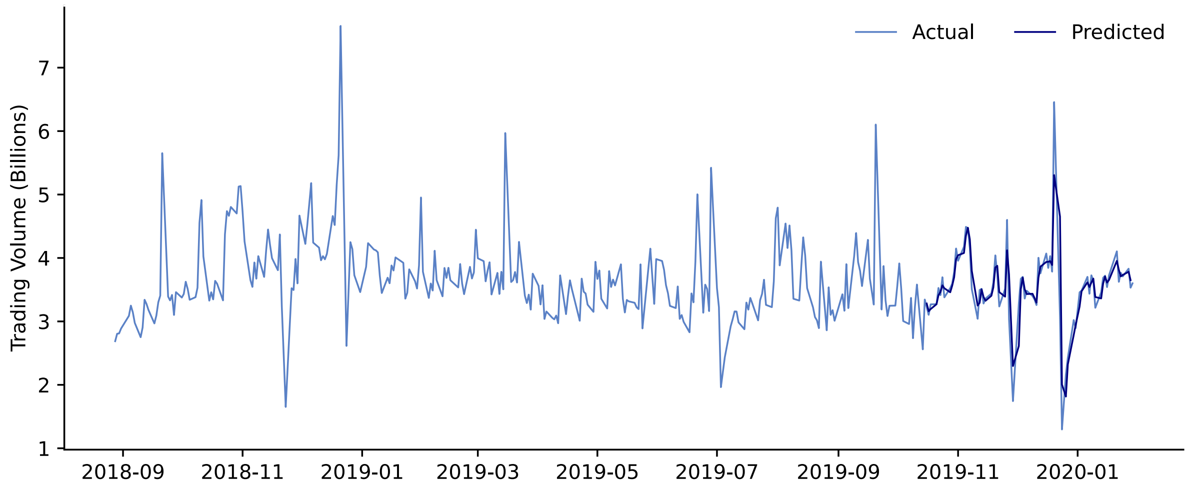


Fig 100: GSPC volume prediction in bear market.

A.2.2.2 Multi-Task Deep Learning Model

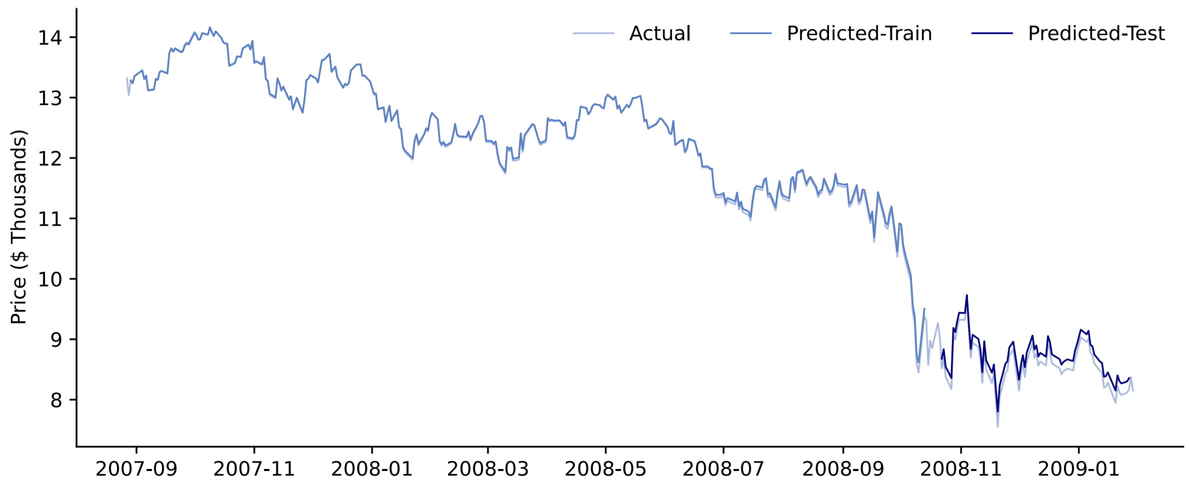


Fig 101: DJI price prediction in bear market, GRU model.

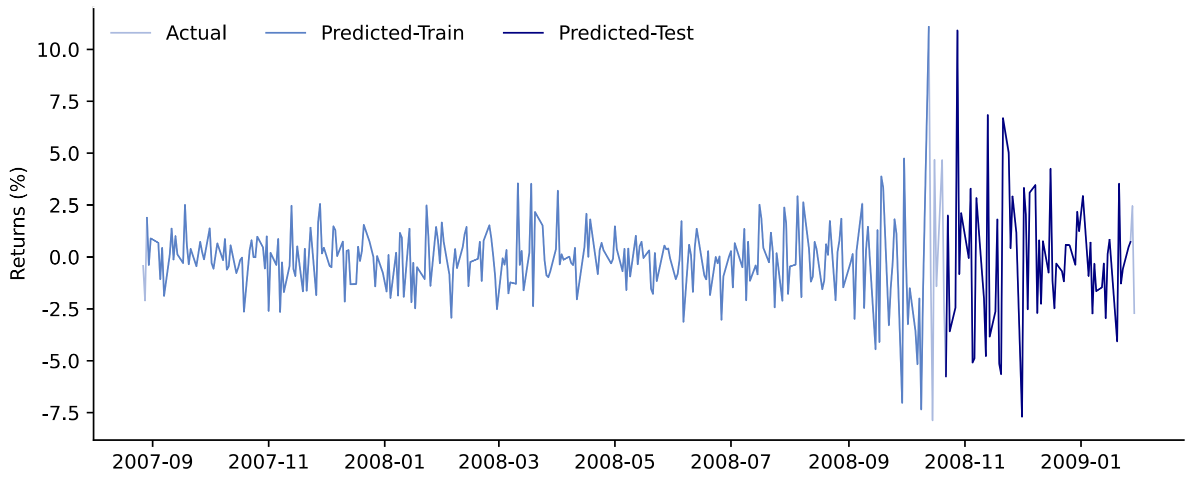


Fig 102: DJI returns prediction in bear market, GRU model.

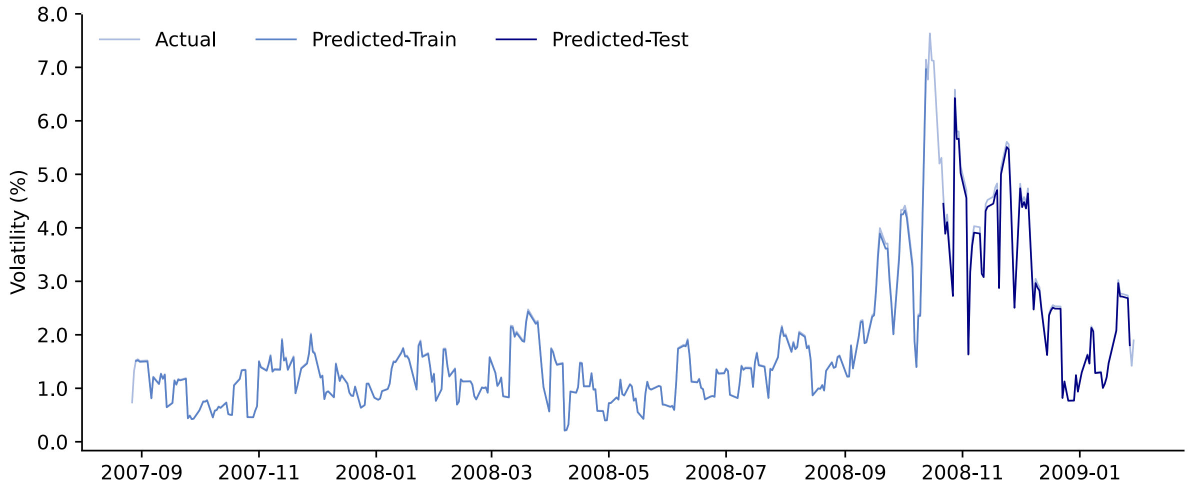


Fig 103: DJI volatility prediction in bear market, GRU model.

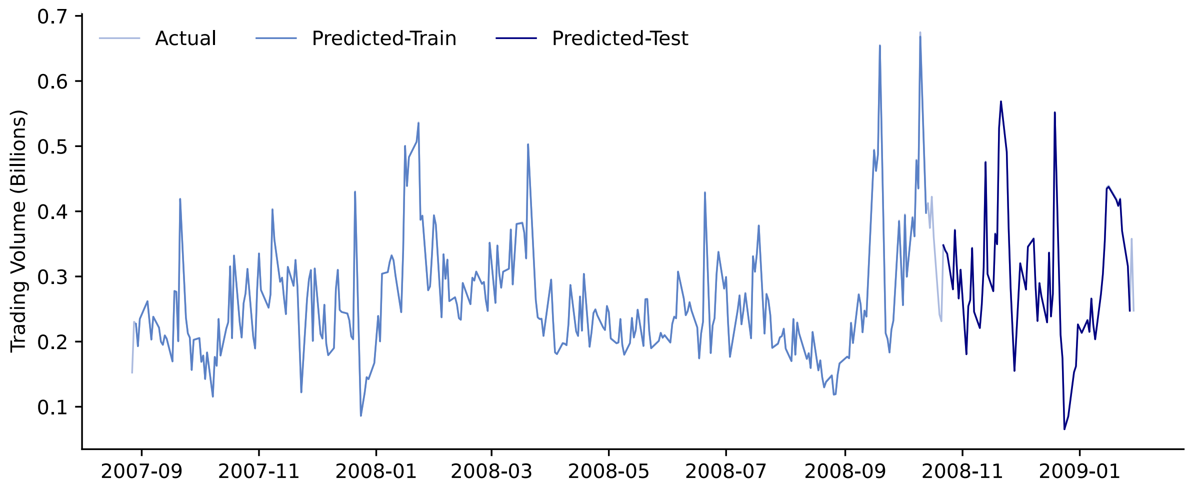


Fig 104: DJI trading volume prediction in bear market, GRU model.

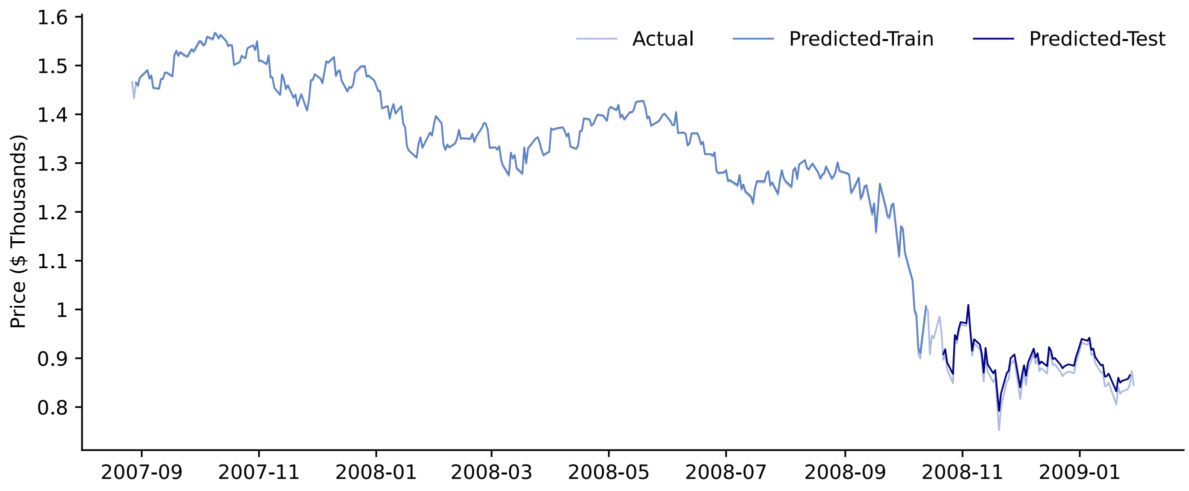


Fig 105: GSPC price prediction in bear market, GRU model.

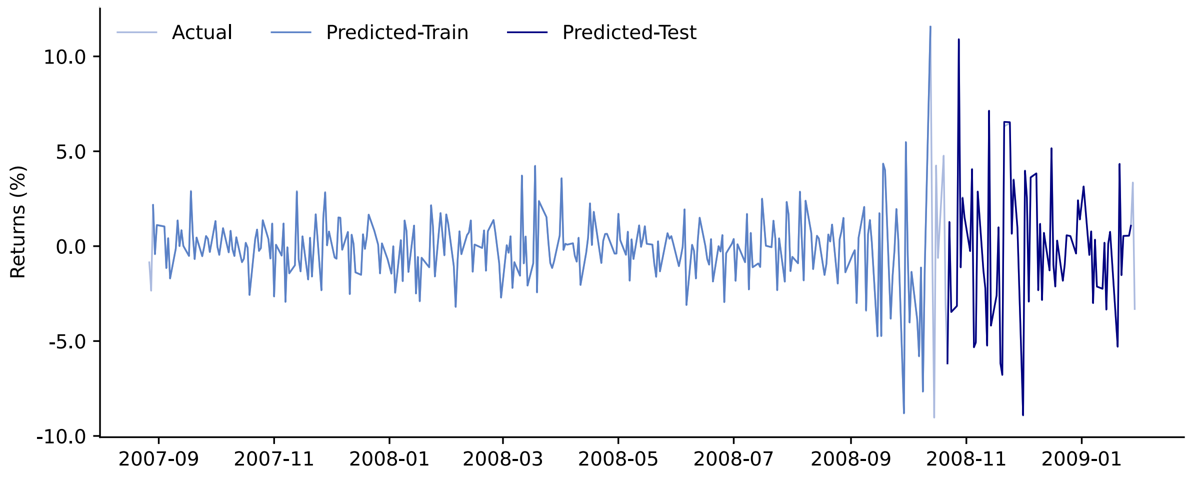


Fig 106: GSPC returns prediction in bear market, GRU model.

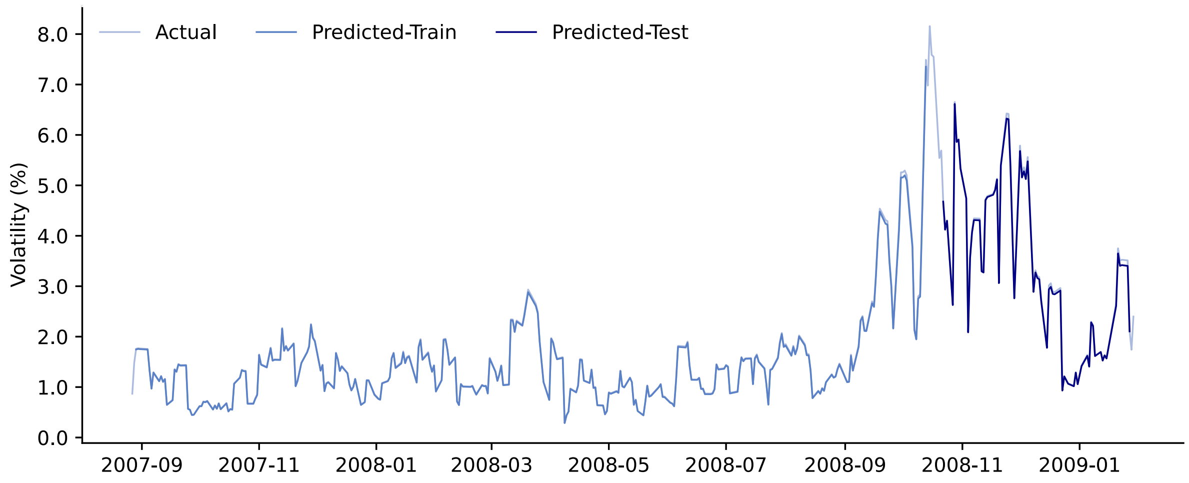


Fig 107: GSPC volatility prediction in bear market, GRU model.

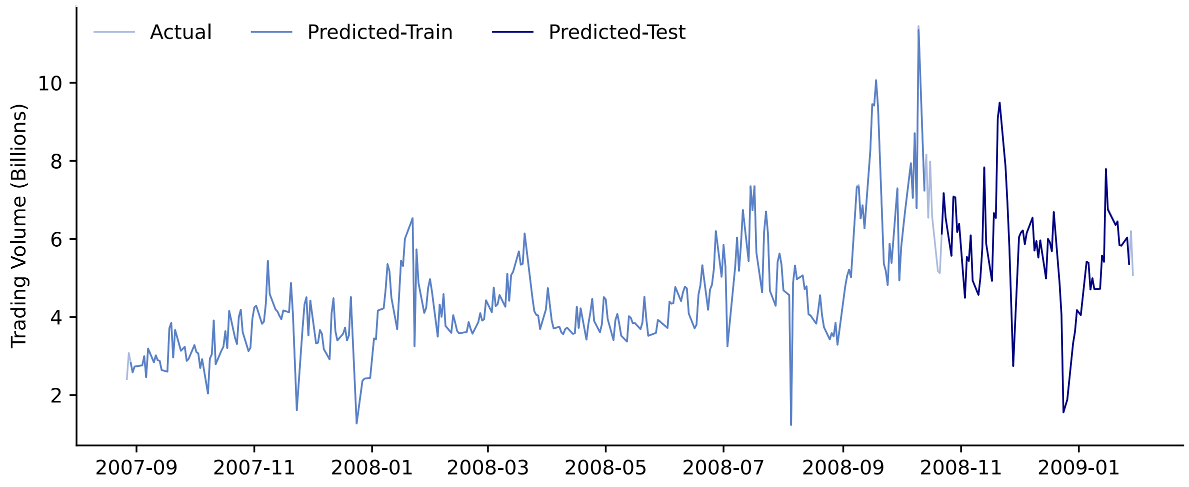


Fig 108: GSPC trading volume prediction in bear market, GRU model.

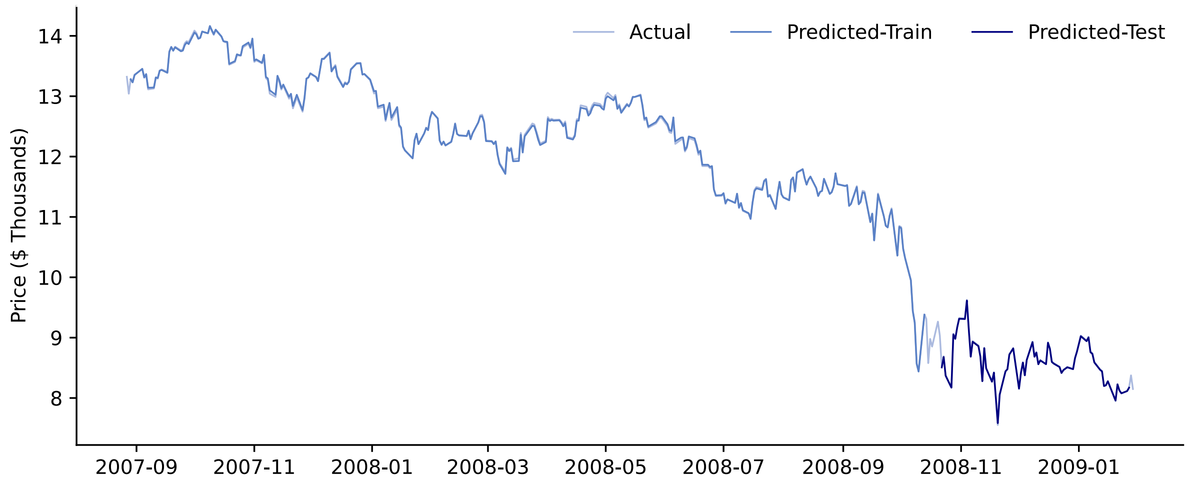


Fig 109: DJI price prediction in bear market, LSTM model.

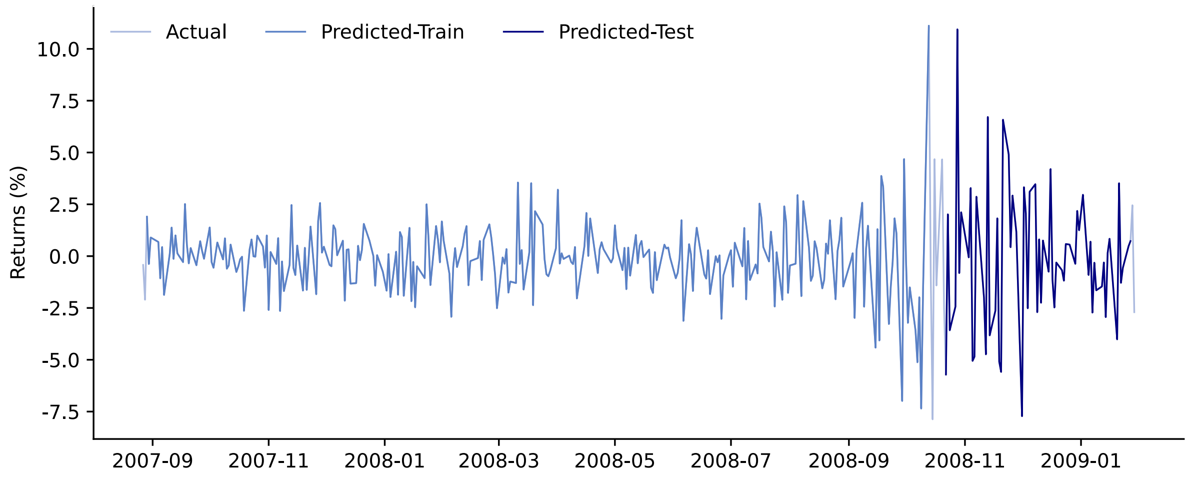


Fig 110: DJI returns prediction in bear market, LSTM model.

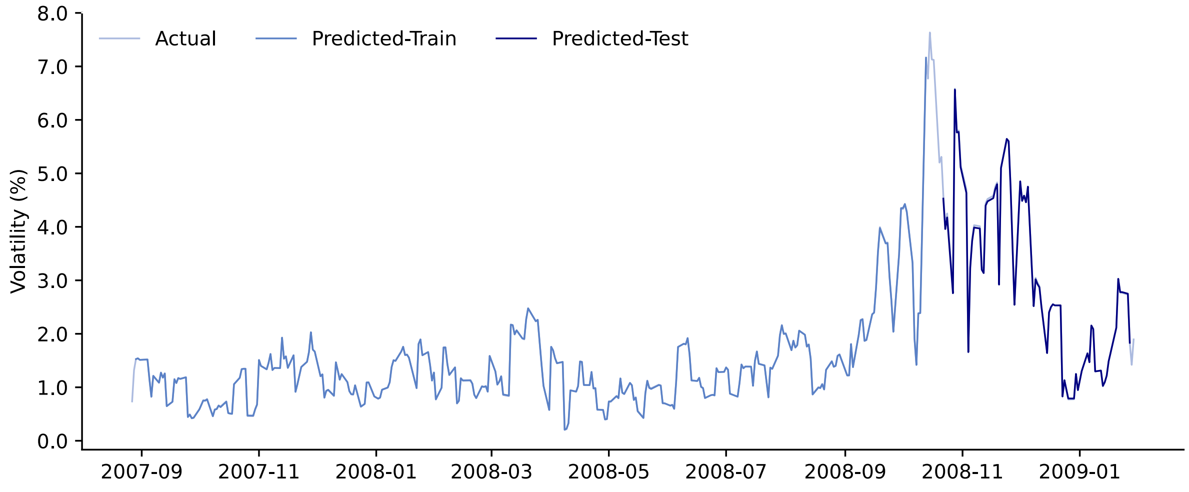


Fig 111: DJI volatility prediction in bear market, LSTM model.

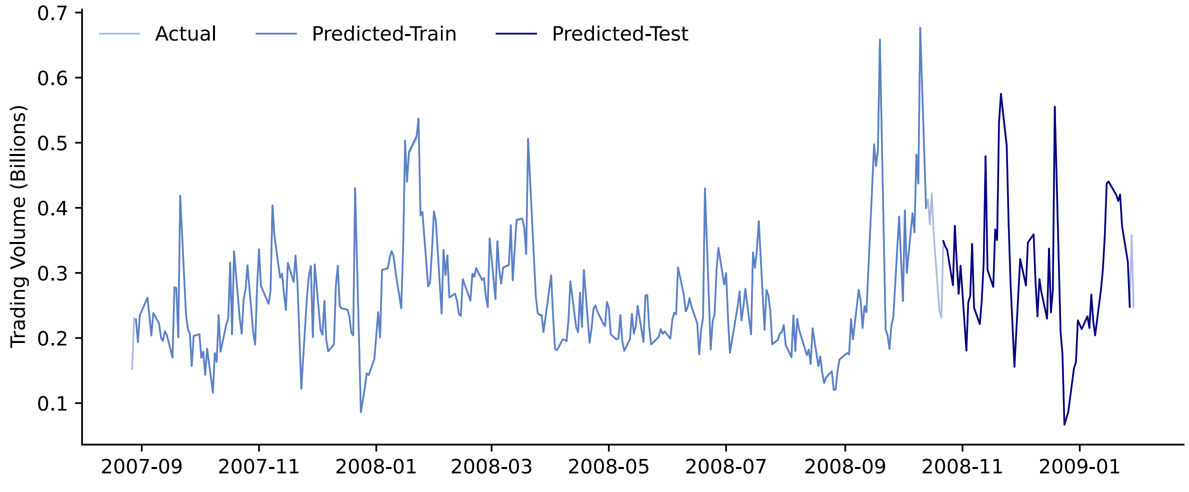


Fig 112: DJI trading volume prediction in bear market, LSTM model.

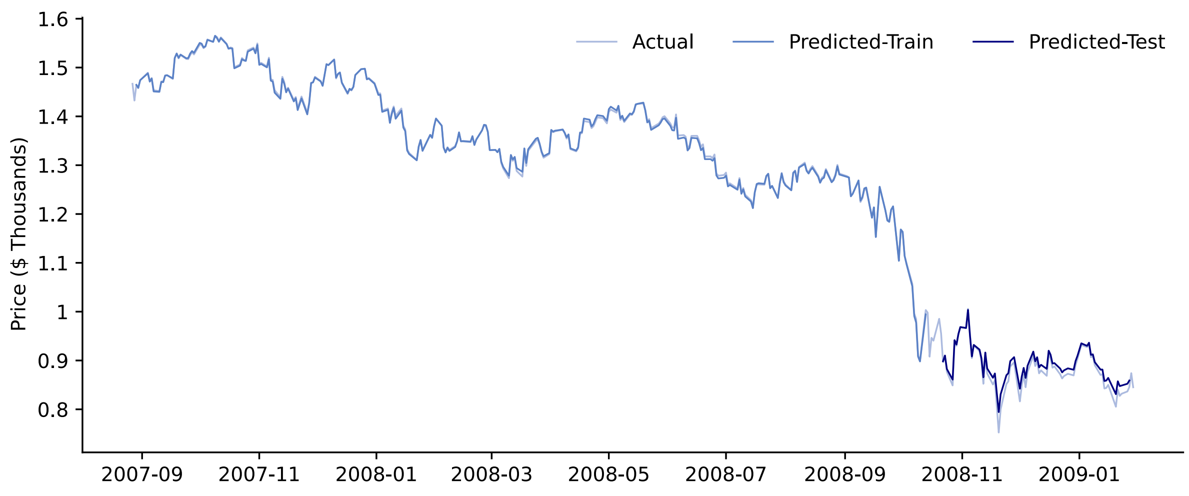


Fig 113: GSPC price prediction in bear market, LSTM model.

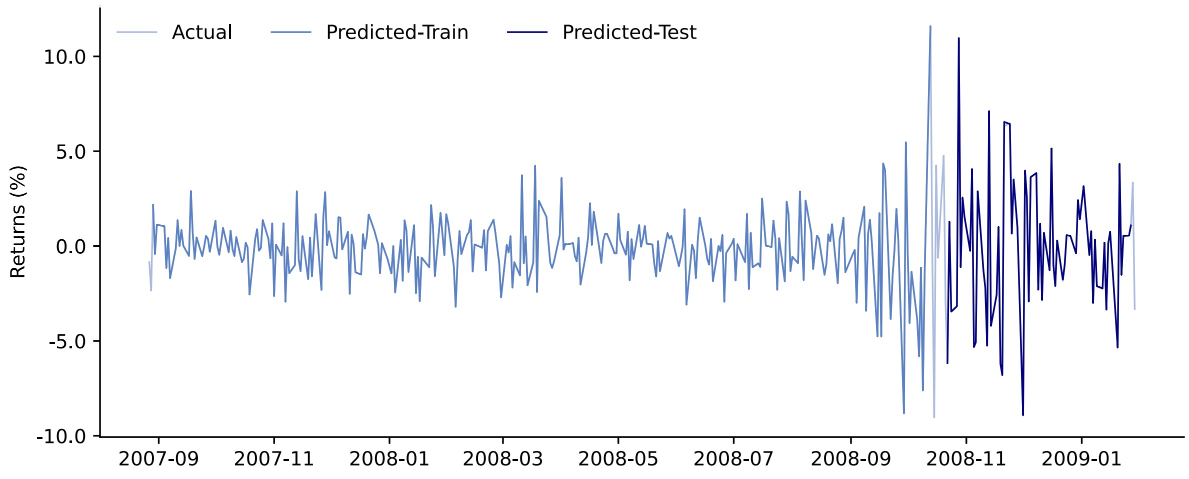


Fig 114: GSPC returns prediction in bear market, LSTM model.

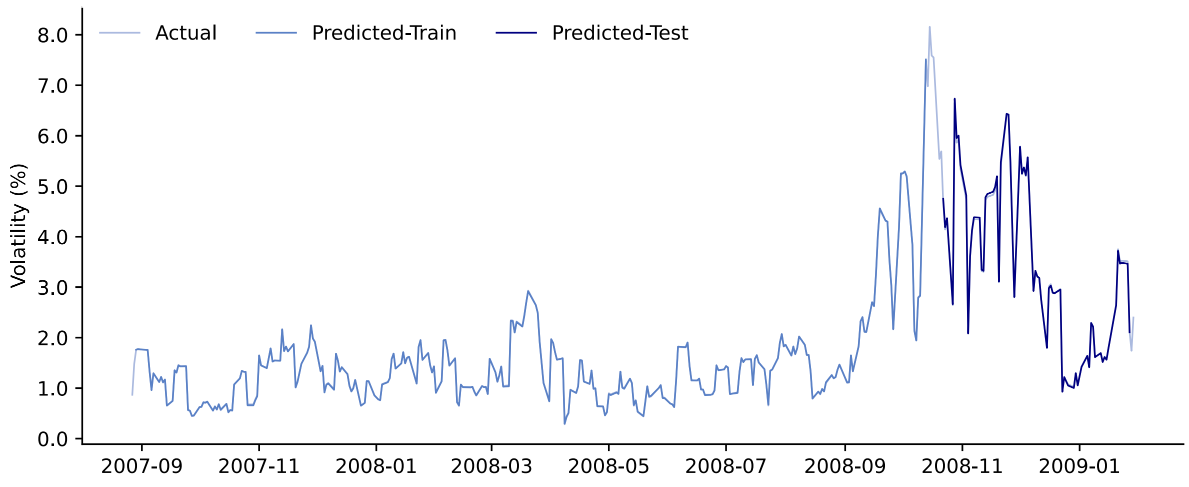


Fig 115: GSPC volatility prediction in bear market, LSTM model.

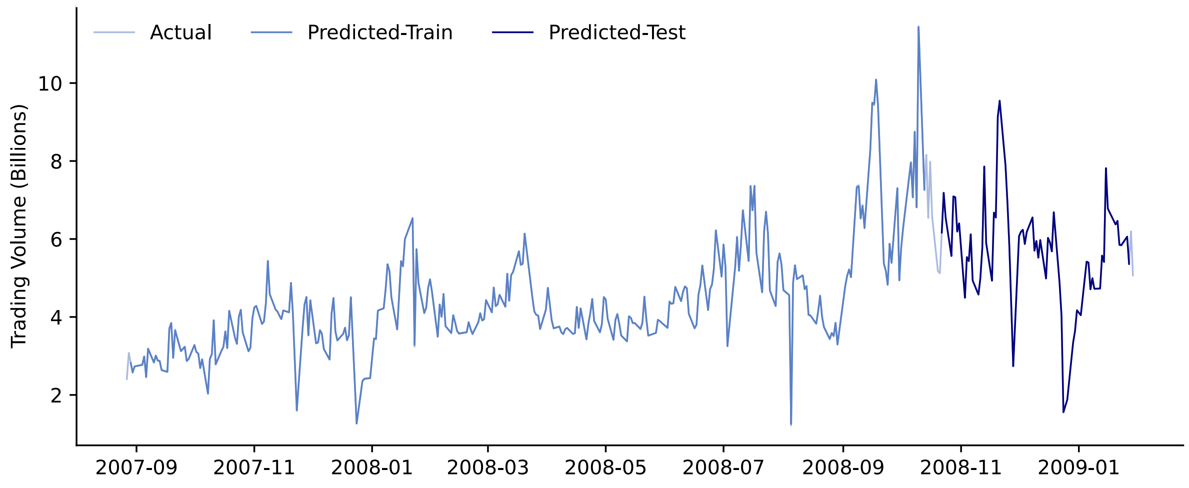


Fig 116: GSPC trading volume prediction in bear market, LSTM model.

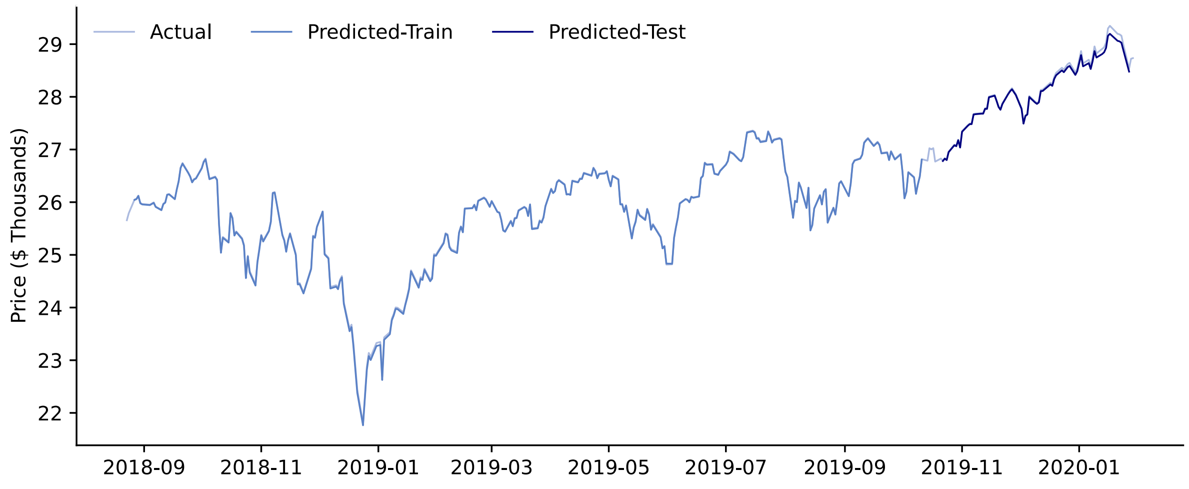


Fig 117: DJI price prediction in bull market, GRU model.

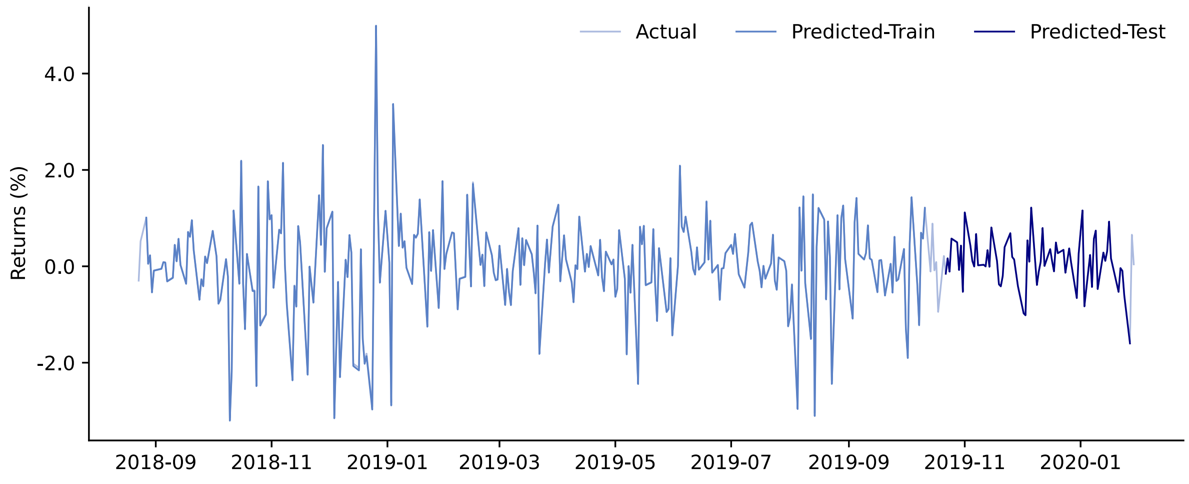


Fig 118: DJI returns prediction in bull market, GRU model.

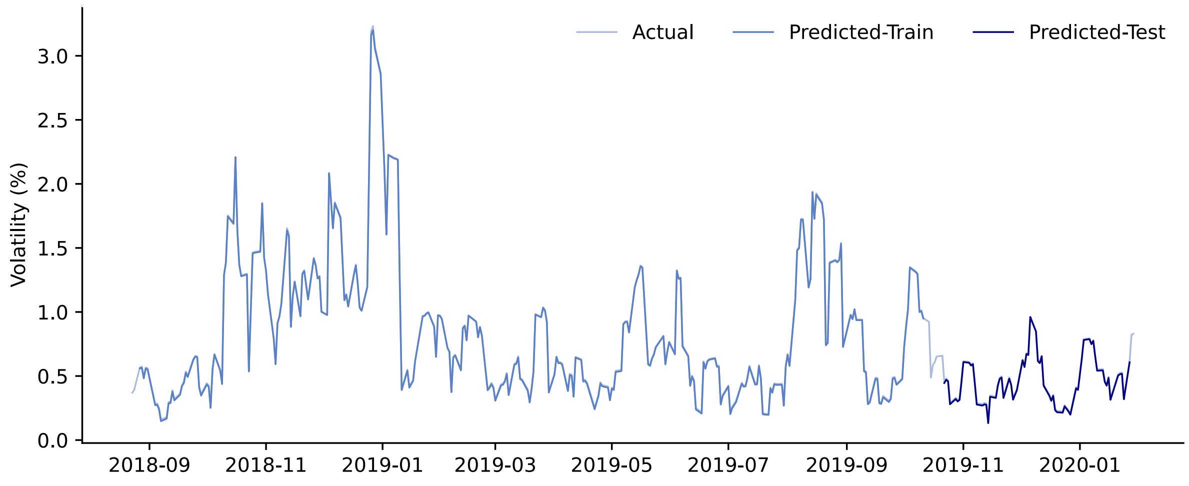


Fig 119: DJI volatility prediction in bull market, GRU model.

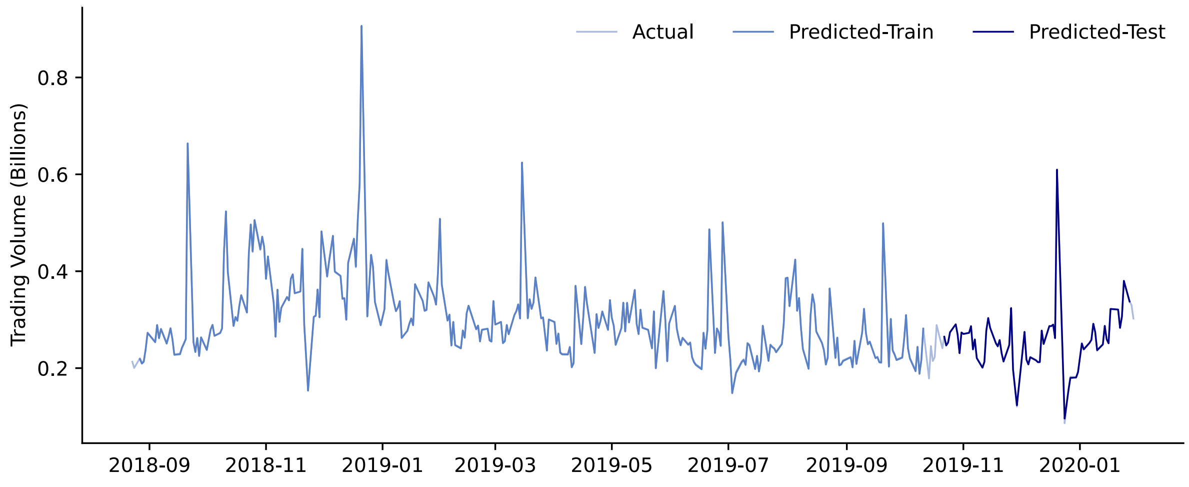


Fig 120: DJI trading volume prediction in bull market, GRU model.

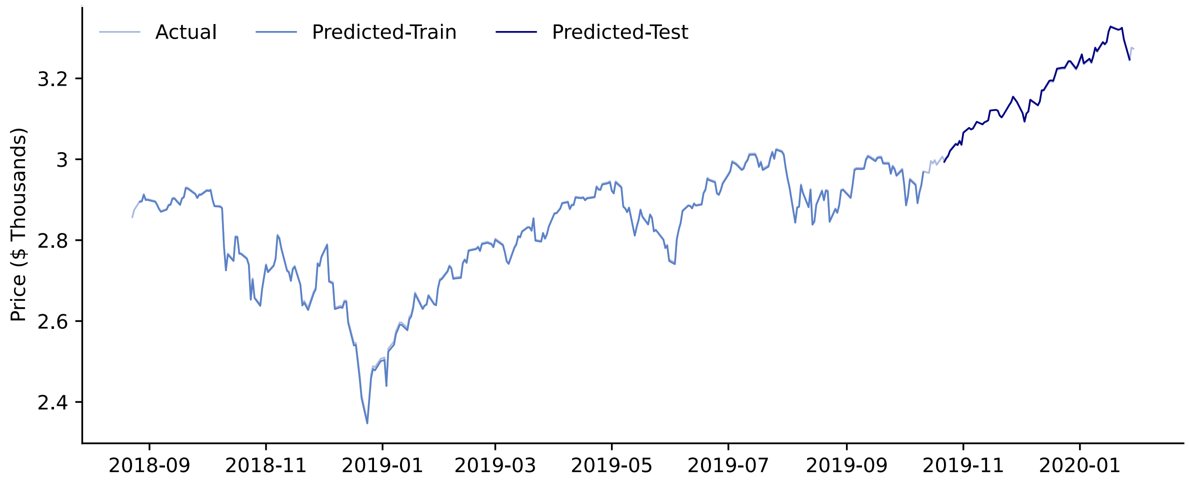


Fig 121: GSPC price prediction in bull market, GRU model.

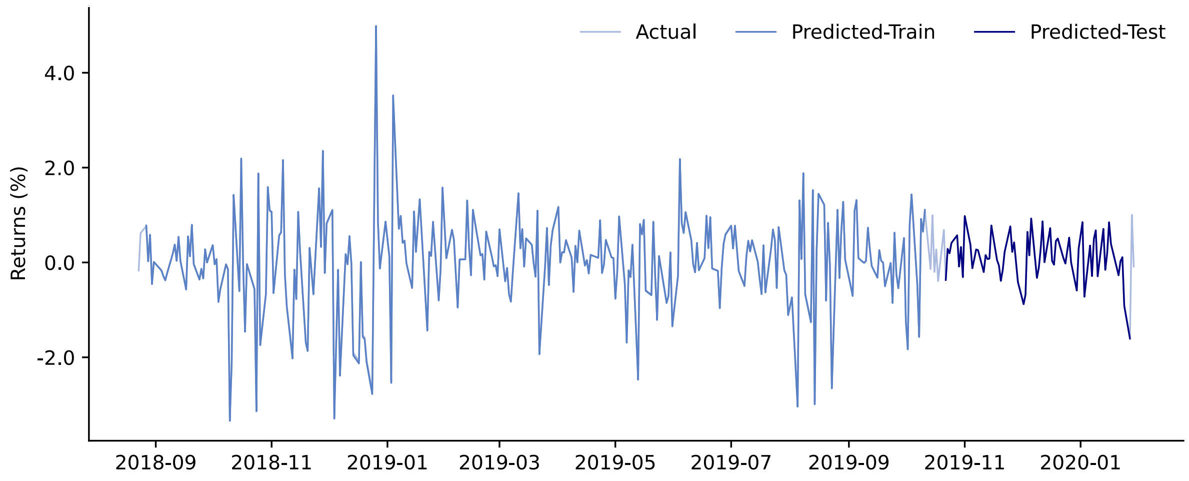


Fig 122: GSPC returns prediction in bull market, GRU model.

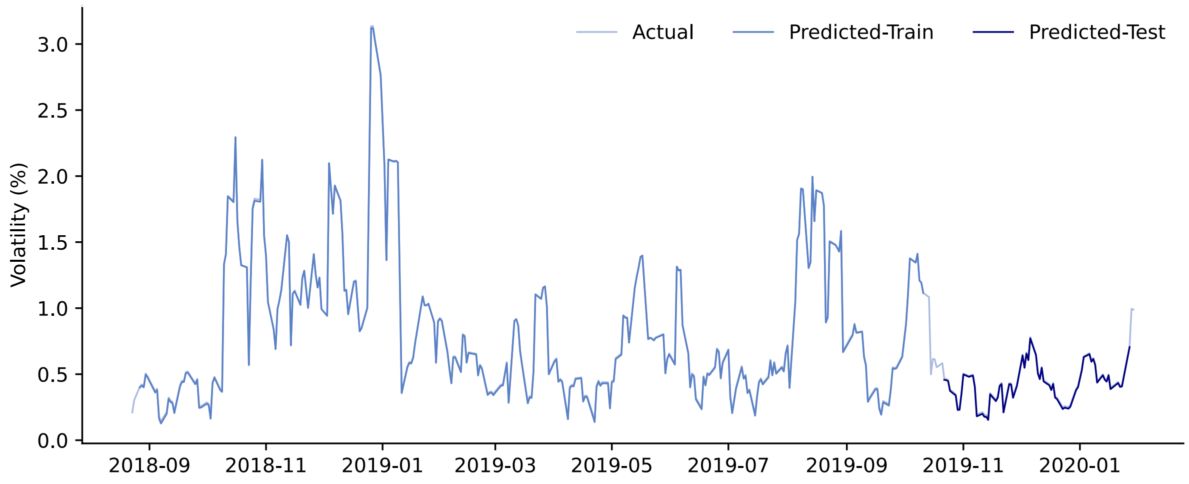


Fig 123: GSPC volatility prediction in bull market, GRU model.

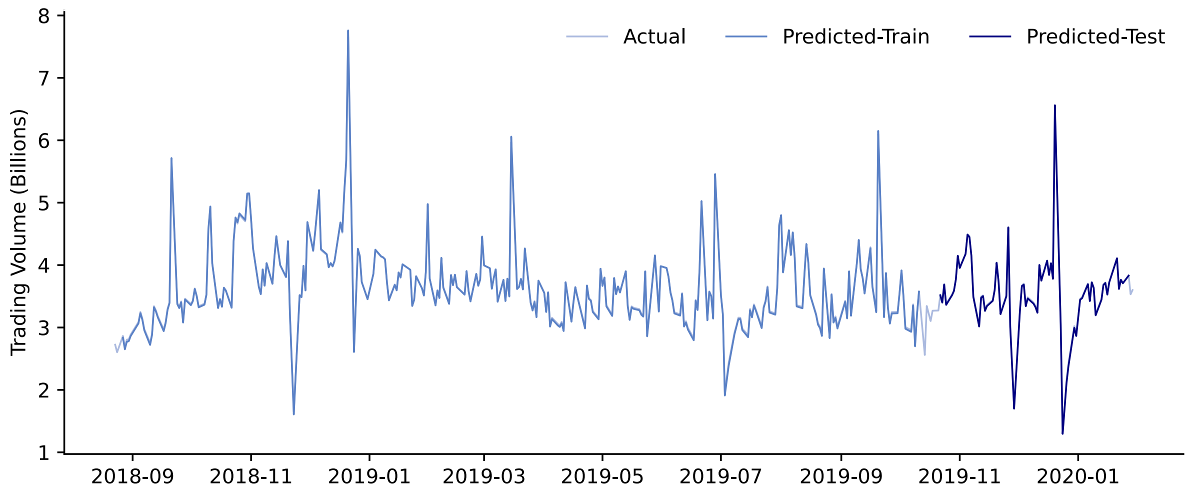


Fig 124: GSPC trading volume prediction in bull market, GRU model.

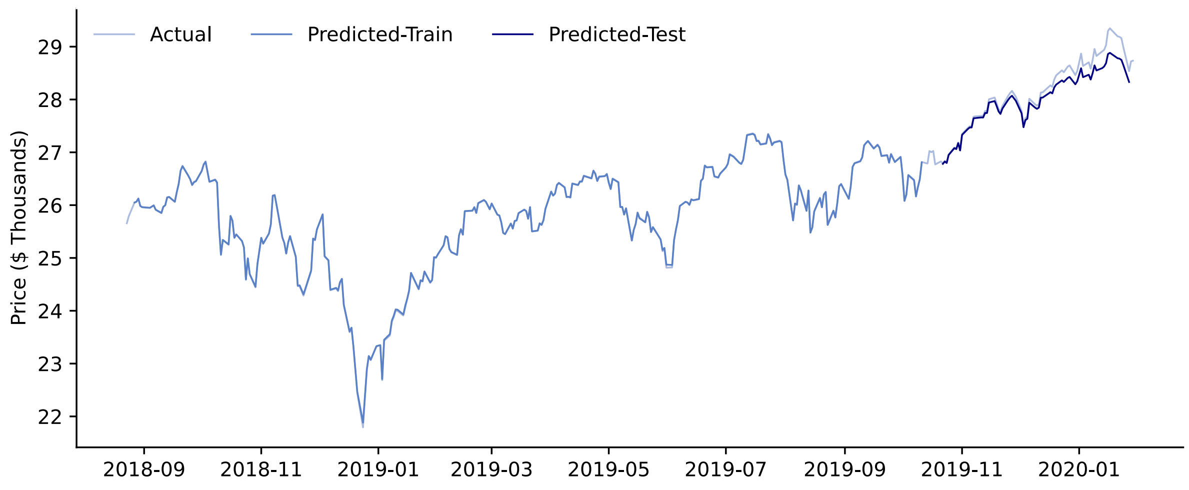


Fig 125: DJI price prediction in bull market, LSTM model.

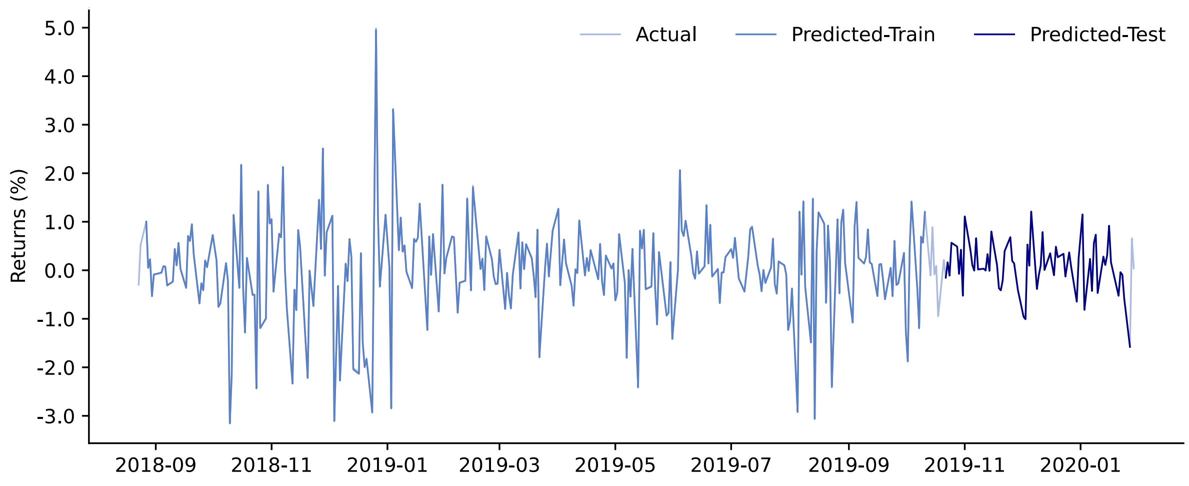


Fig 126: DJI returns prediction in bull market, LSTM model.

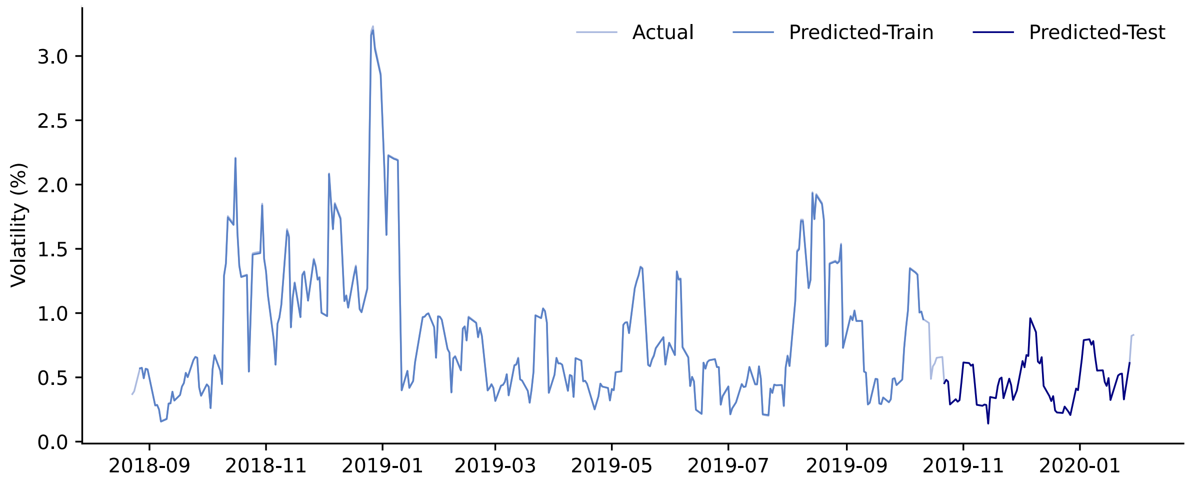


Fig 127: DJI volatility prediction in bull market, LSTM model.

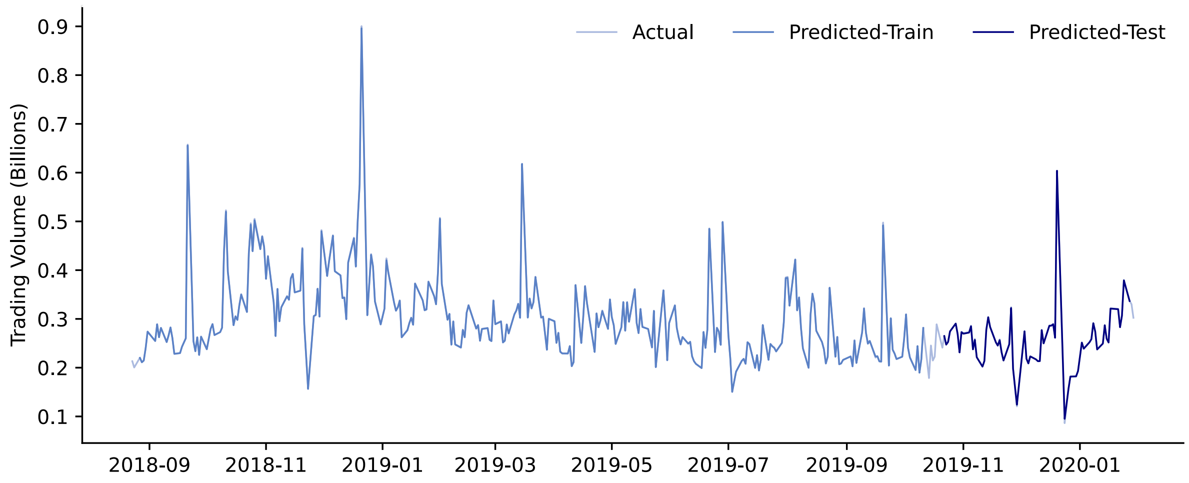


Fig 128: DJI trading volume prediction in bull market, LSTM model.

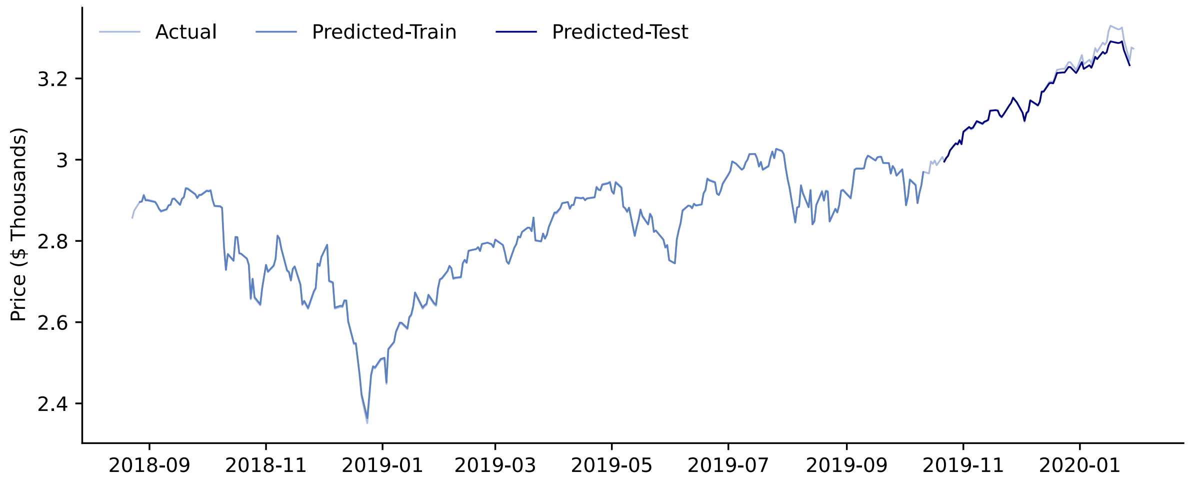


Fig 129: GSPC price prediction in bull market, LSTM model.

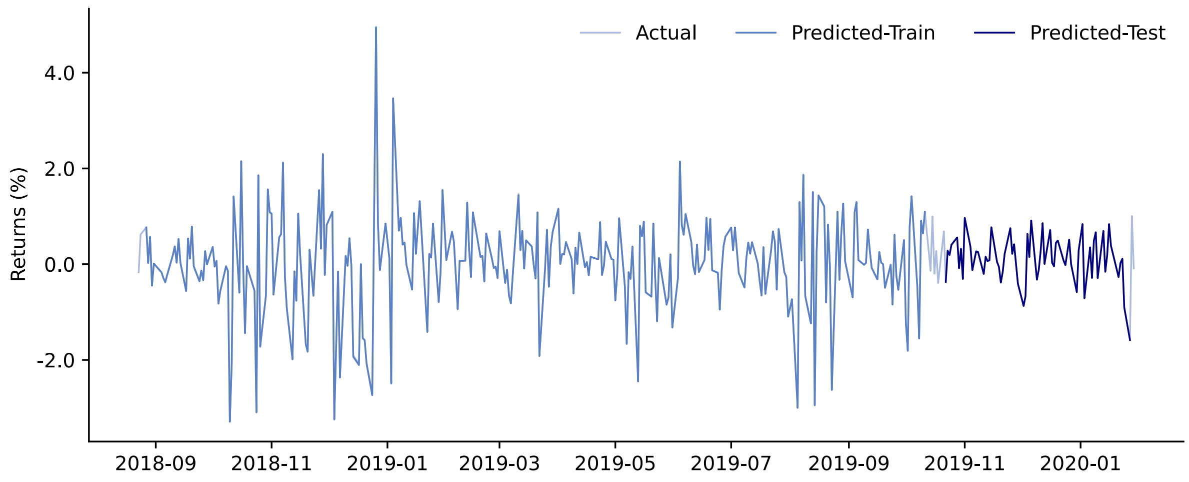


Fig 130: GSPC returns prediction in bull market, LSTM model.

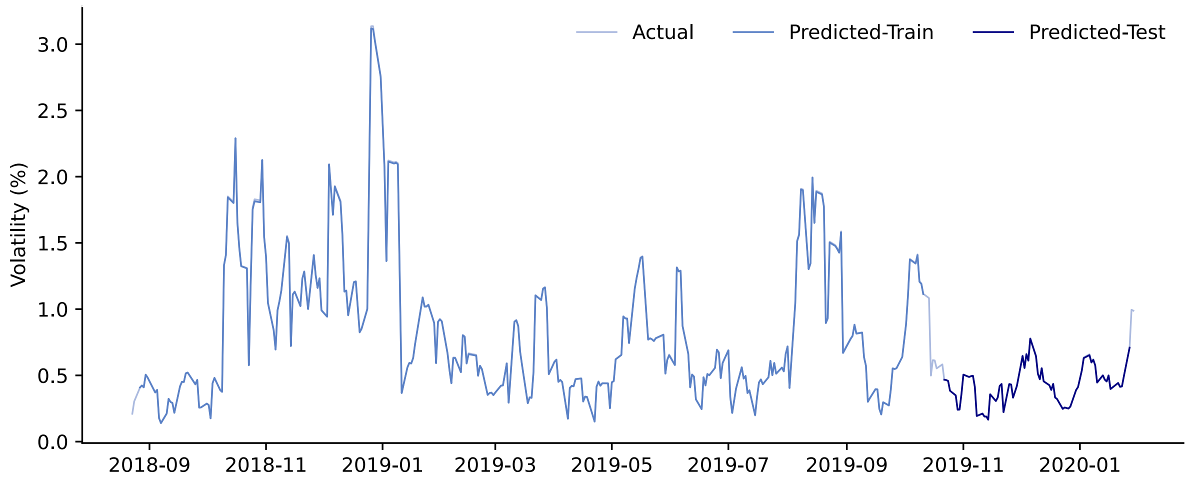


Fig 131: GSPC volatility prediction in bull market, LSTM model.

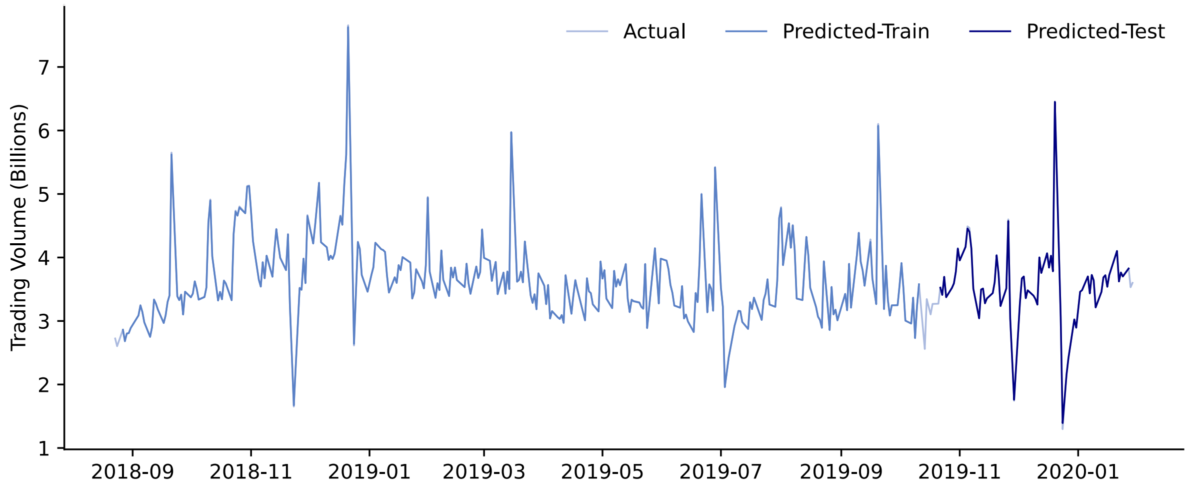


Fig 132: GSPC trading volume prediction in bull market, LSTM model.

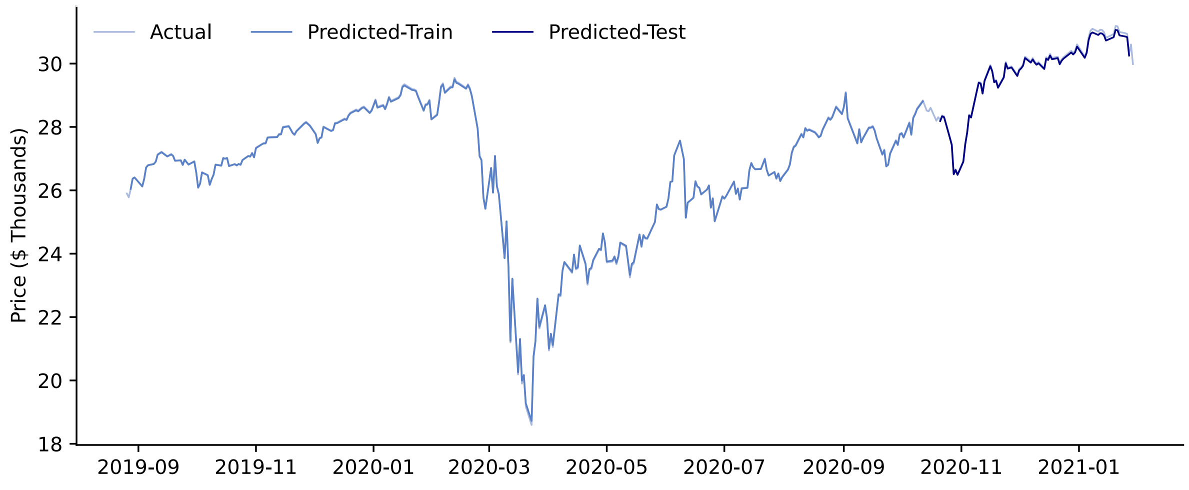


Fig 133: DJI price prediction in volatile market, GRU model.

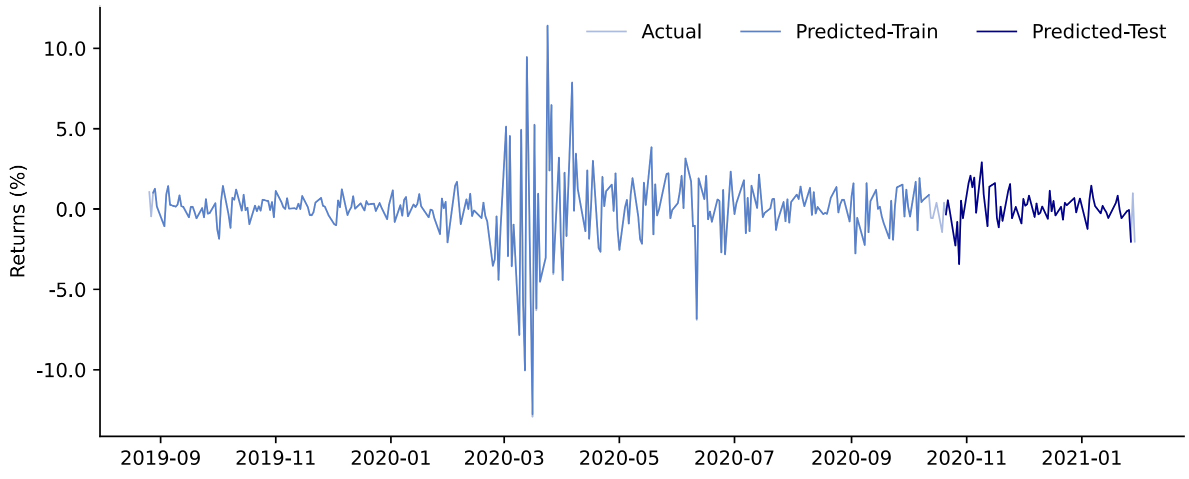


Fig 134: DJI returns prediction in volatile market, GRU model.

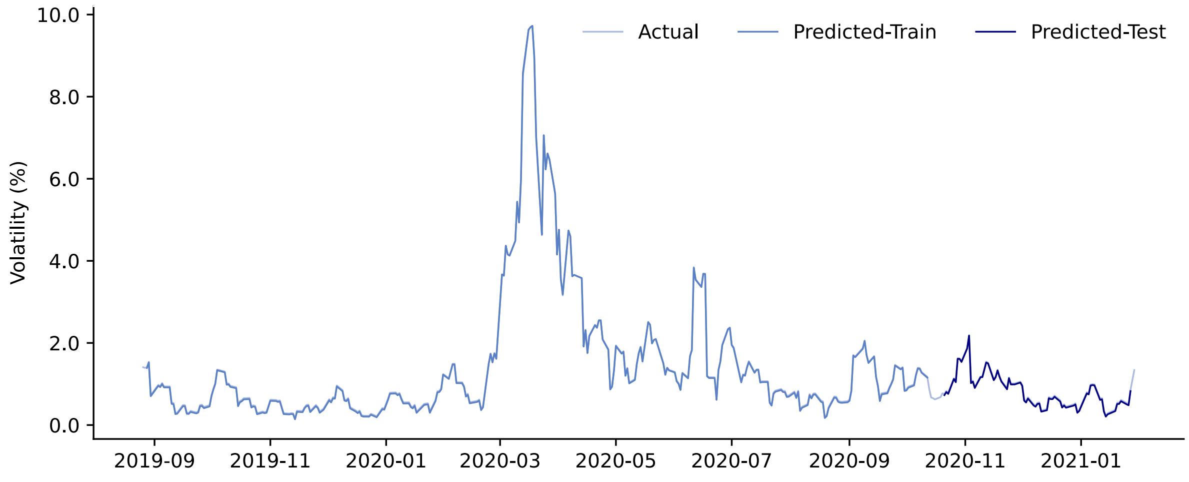


Fig 135: DJI volatility prediction in volatile market, GRU model.

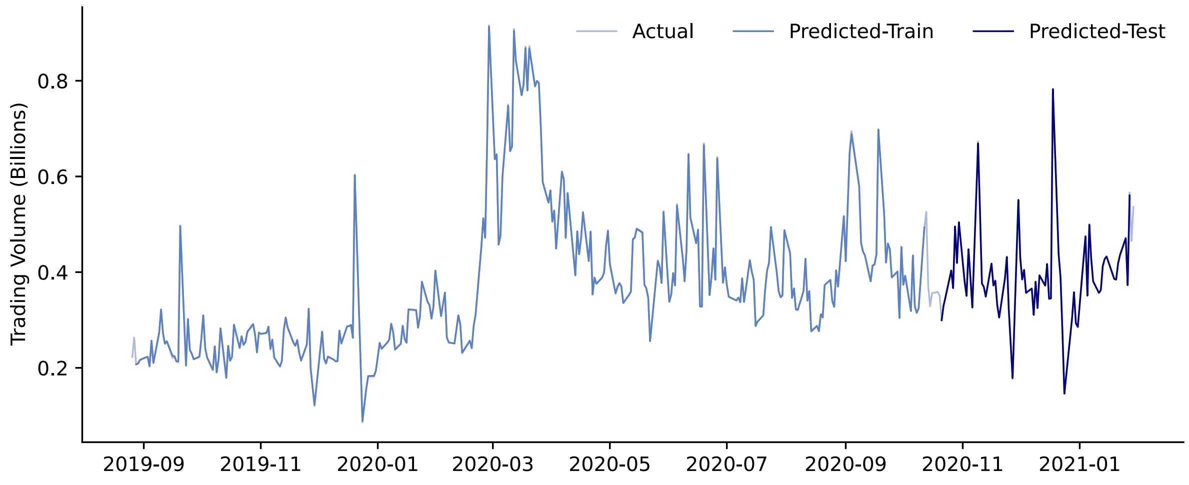


Fig 136: DJI trading volume prediction in volatile market, GRU model.

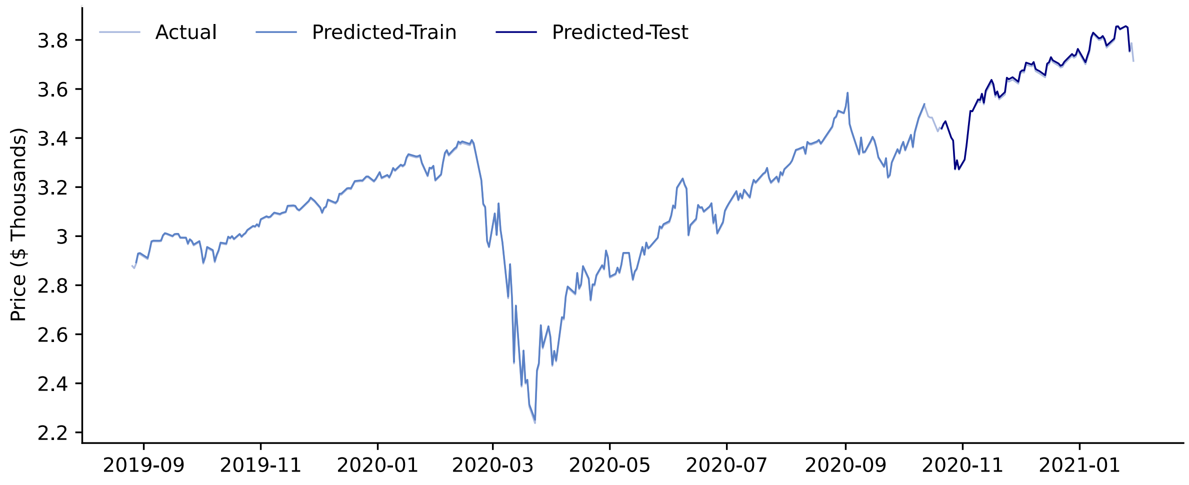


Fig 137: GSPC price prediction in volatile market, GRU model.

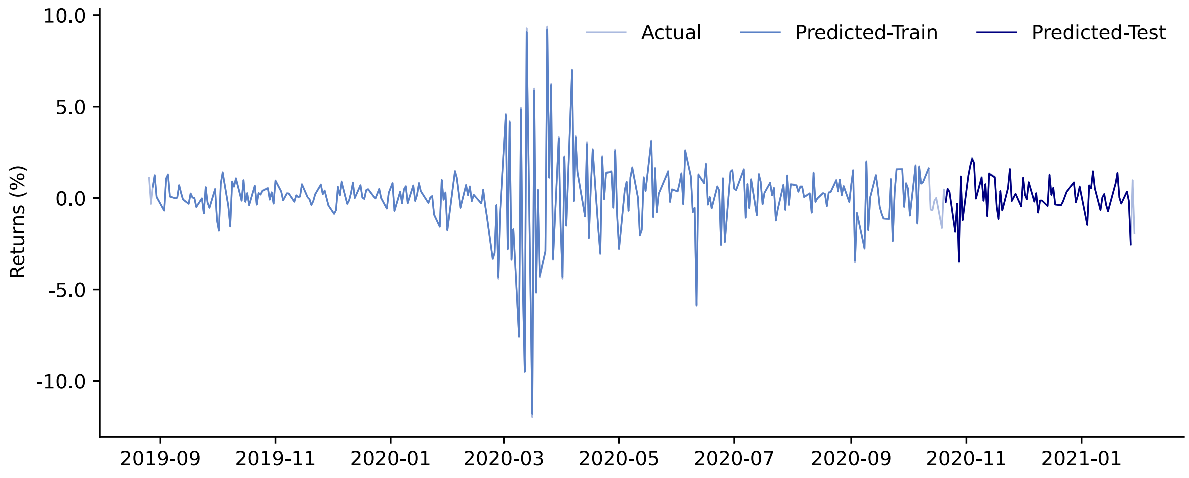


Fig 138: GSPC returns prediction in volatile market, GRU model.

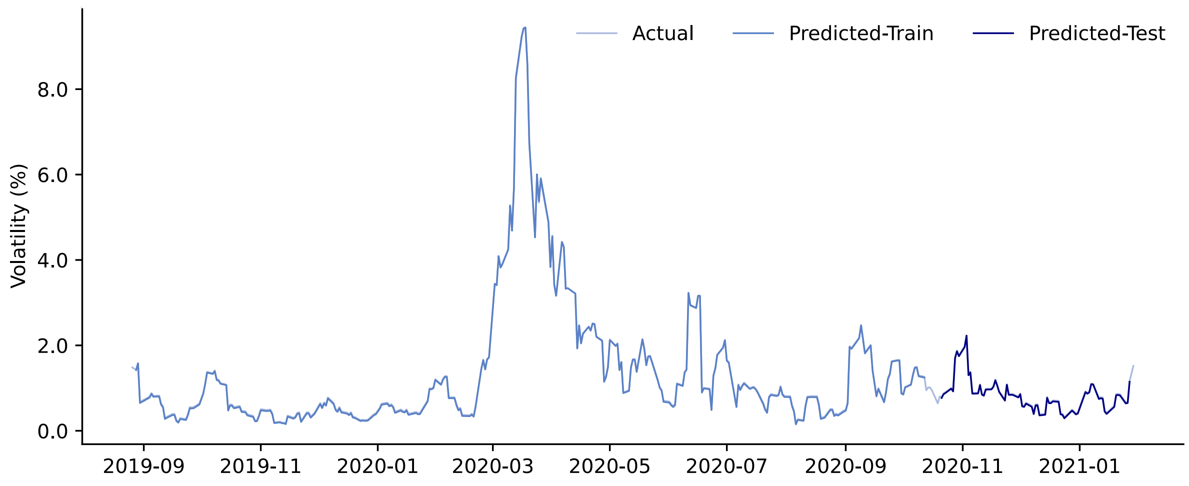


Fig 139: GSPC volatility prediction in volatile market, GRU model.

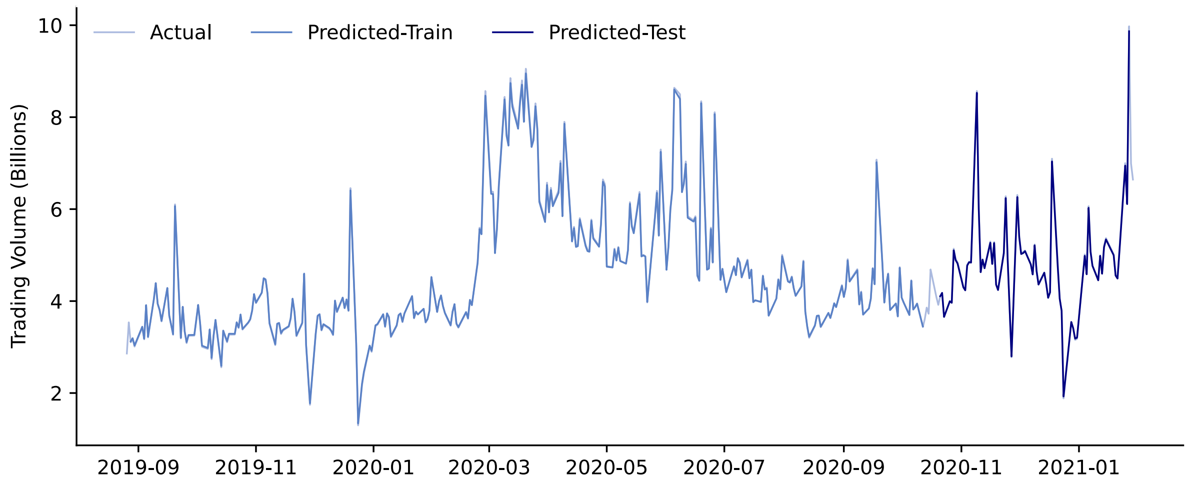


Fig 140: GSPC trading volume prediction in volatile market, GRU model.

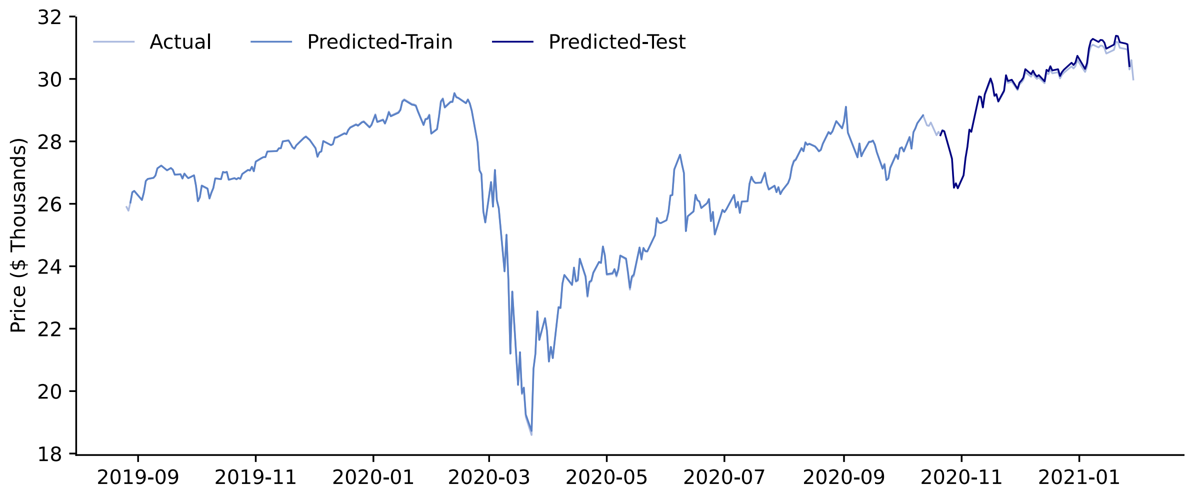


Fig 141: DJI price prediction in volatile market, LSTM model.

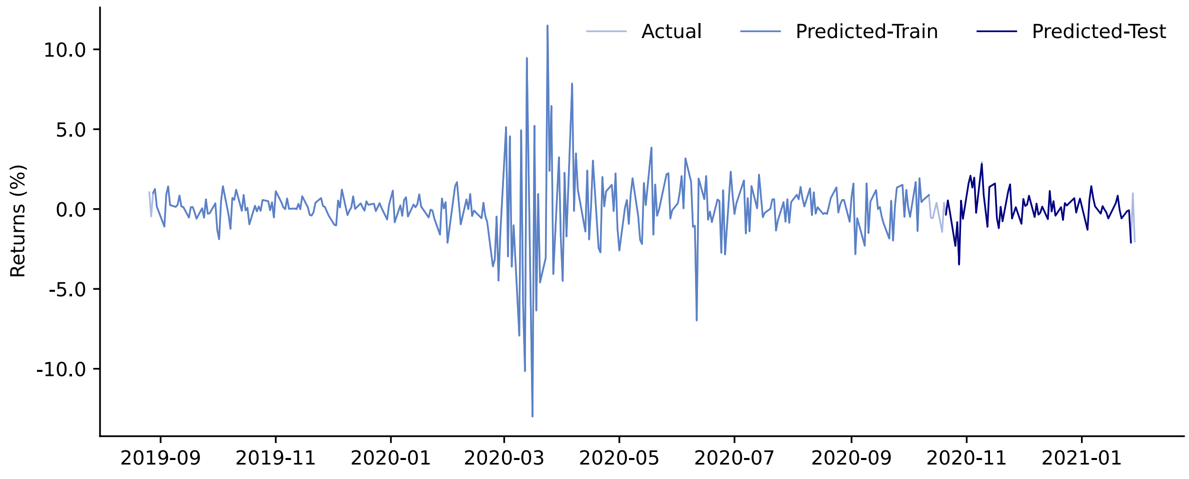


Fig 142: DJI returns prediction in volatile market, LSTM model.

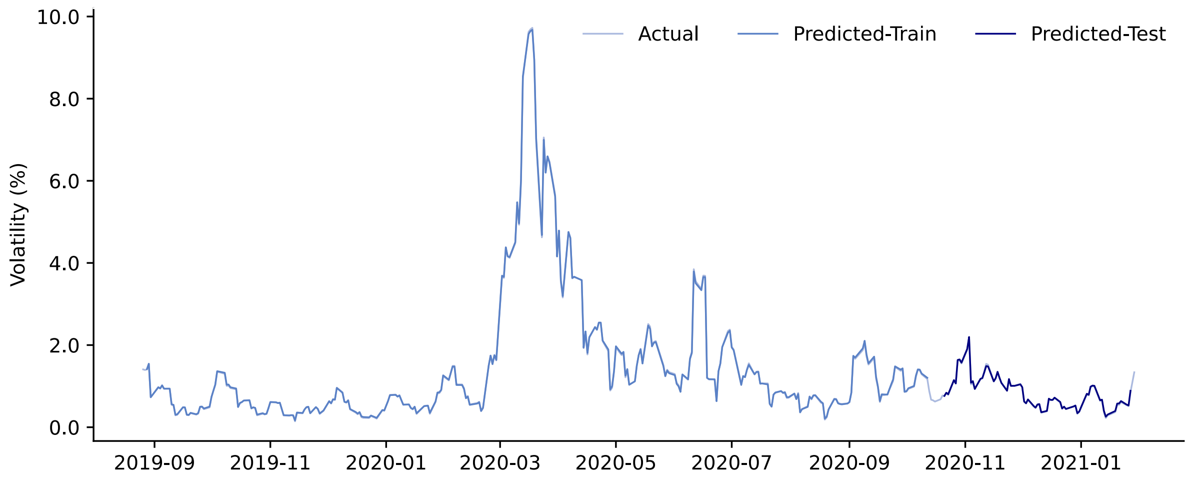


Fig 143: DJI volatility prediction in volatile market, LSTM model.

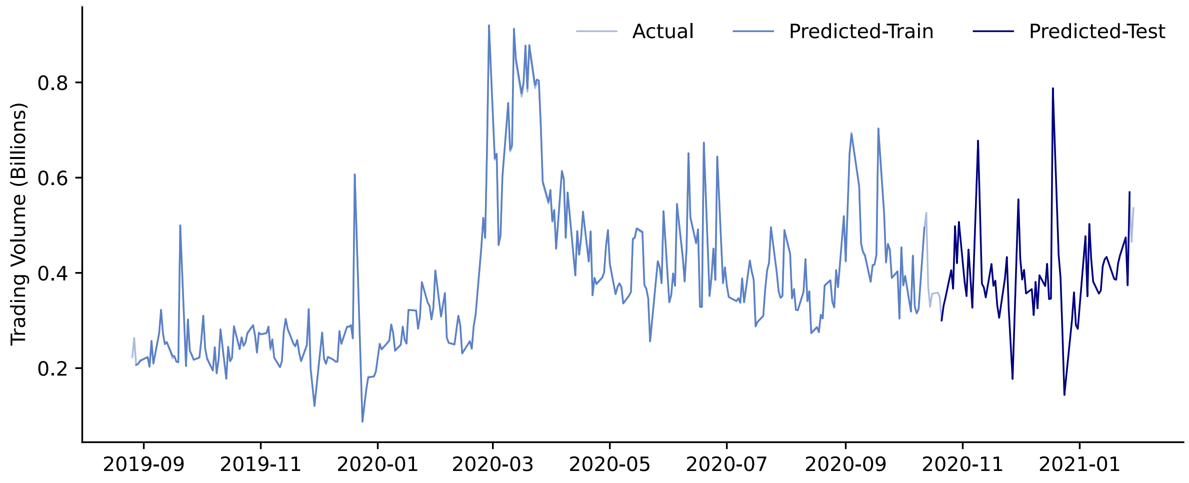


Fig 144: DJI trading volume prediction in volatile market, LSTM model.

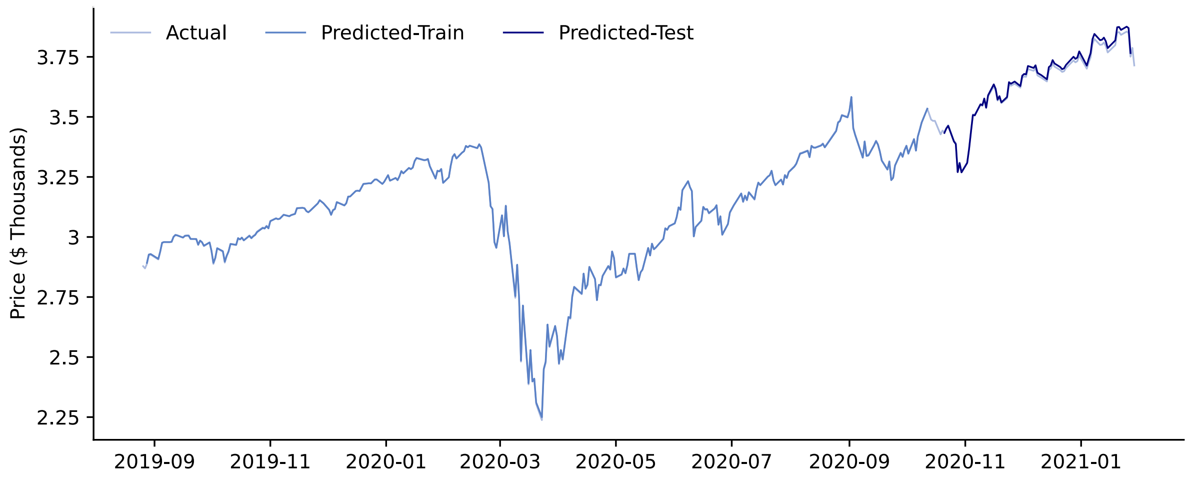


Fig 145: GSPC price prediction in volatile market, LSTM model.

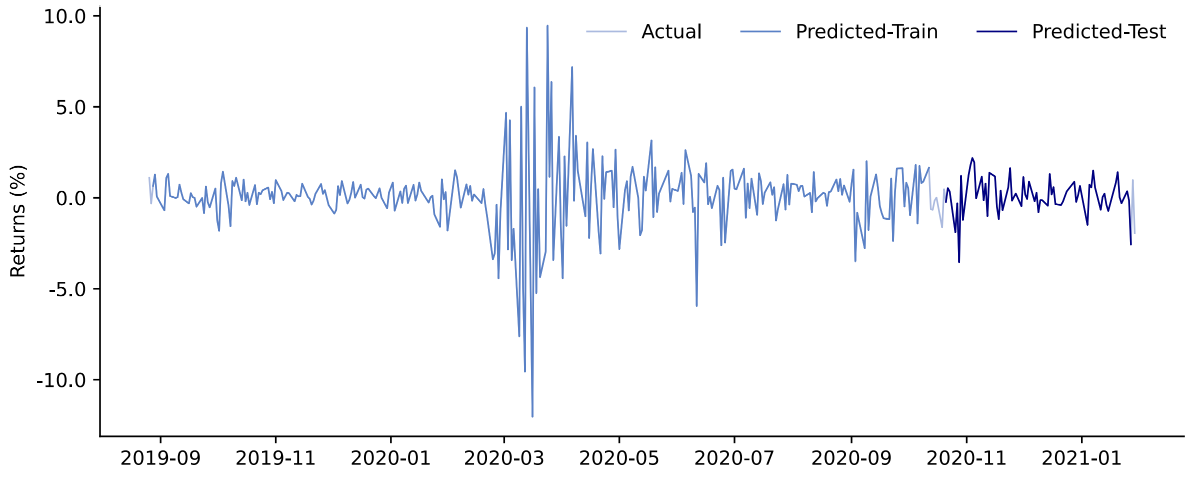


Fig 146: GSPC returns prediction in volatile market, LSTM model.

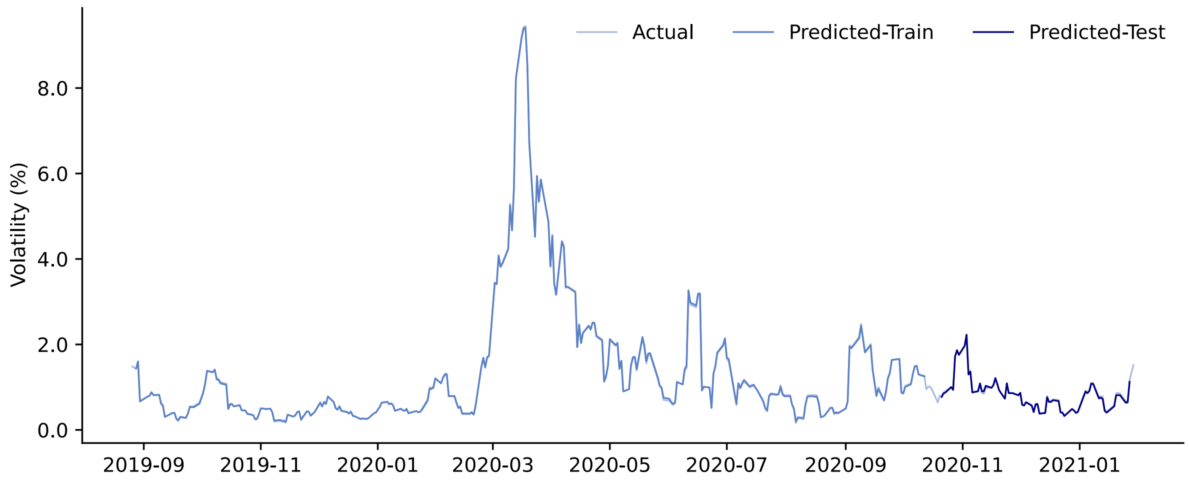


Fig 147: GSPC volatility prediction in volatile market, LSTM model.

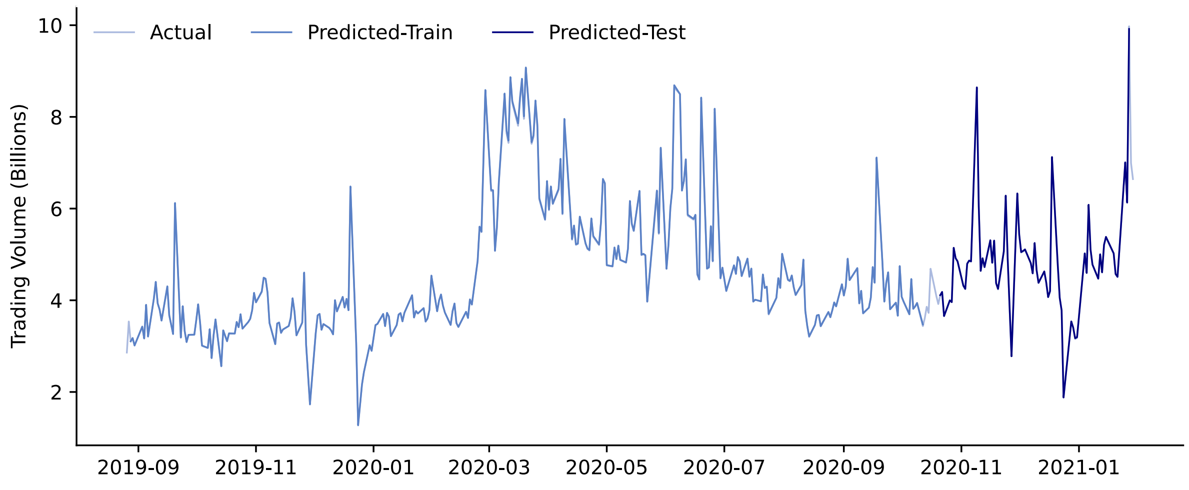


Fig 148: GSPC trading volume prediction in volatile market, LSTM model.

Appendix B
Deep Learning Model Optimization

This appendix serves as a valuable resource for understanding the intricacies of performance optimization strategies deployed throughout this research. These methodologies played a pivotal role in the fine-tuning of hyperparameters for the tests conducted in both Chapter 5 and Chapter 6. The optimization efforts were twofold: one set aimed at enhancing performance to meet benchmark testing standards, while the other set aimed at deliberately degrading prediction performance to facilitate the testing of diverse deep learning implementations.

The mathematical expressions and formulas used for these calculations can be found in Equation 7.1, providing a comprehensive reference for the methods applied during the tuning process.

<u>Single input/ Single Output</u>

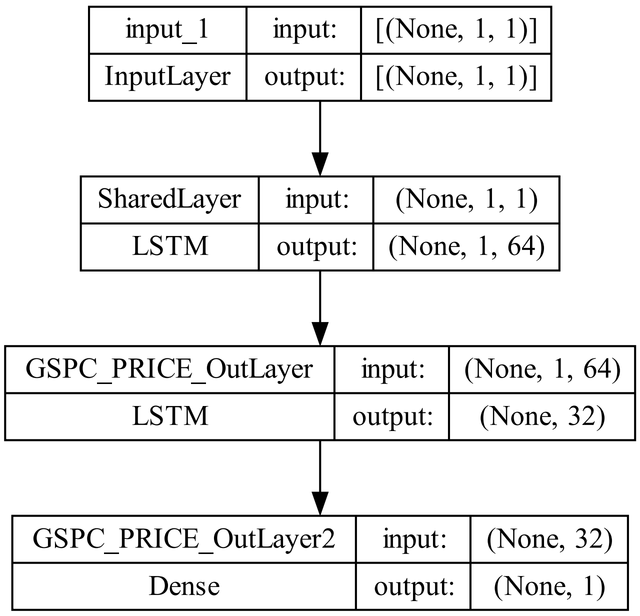


Figure 149: DL model with single input/output.

```
Layer (type)                         Output Shape          Param #
==================================================================
input_1 (InputLayer)                 [(None, 1, 1)]        0

SharedLayer (LSTM)                   (None, 1, 64)         16896

GSPC_PRICE_OutLayer (LSTM)           (None, 32)            12416

GSPC_PRICE_OutLayer2 (Dense          (None, 1)             33
)

==================================================================
Total params: 29,345
Trainable params: 29,345
Non-trainable params: 0
```

Figure 150: Single-Input DL model summary.

```
model.weights[0].shape: TensorShape([1, 256])

model.weights[1].shape: TensorShape([64, 256])

model.weights[2].shape: TensorShape([256])

model.weights[3].shape: TensorShape([64, 128])

model.weights[4].shape: TensorShape([32, 128])

model.weights[5].shape: TensorShape([128])

model.weights[6].shape: TensorShape([32, 1])

model.weights[7].shape: TensorShape([1])
```

param1 (layer1) = 4 ((1 + 64) * 64 + 64) = 16896

param2 (layer2) = 4 ((64 + 32) * 32 + 32) = 12416

param3 = 32 +1 = 33

Total = 16896 + 12416 + 33 = 29345

Multiple input/ Multiple Output

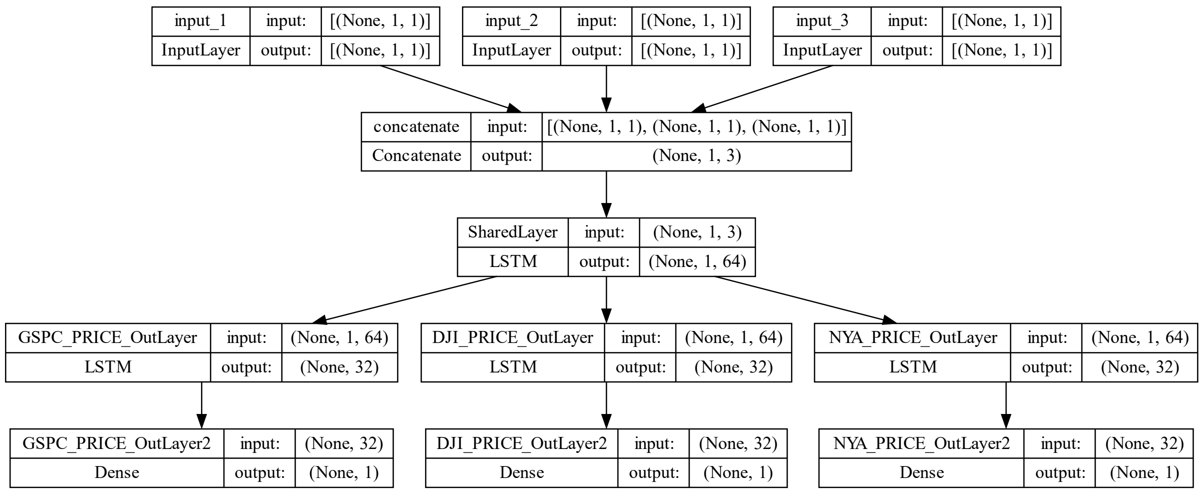

input_1 | input: | [(None, 1, 1)]
InputLayer | output: | [(None, 1, 1)]
input_2 | input: | [(None, 1, 1)]
InputLayer | output: | [(None, 1, 1)]
input_3 | input: | [(None, 1, 1)]
InputLayer | output: | [(None, 1, 1)]
concatenate | input: | [(None, 1, 1), (None, 1, 1), (None, 1, 1)]
Concatenate | output: | (None, 1, 3)
SharedLayer | input: | (None, 1, 3)
LSTM | output: | (None, 1, 64)
GSPC_PRICE_OutLayer | input: | (None, 1, 64)
LSTM | output: | (None, 32)
DJI_PRICE_OutLayer | input: | (None, 1, 64)
LSTM | output: | (None, 32)
NYA_PRICE_OutLayer | input: | (None, 1, 64)
LSTM | output: | (None, 32)
GSPC_PRICE_OutLayer2 | input: | (None, 32)
Dense | output: | (None, 1)
DJI_PRICE_OutLayer2 | input: | (None, 32)
Dense | output: | (None, 1)
NYA_PRICE_OutLayer2 | input: | (None, 32)
Dense | output: | (None, 1)

Figure 151: Multi-Task DL model with multiple inputs and outputs.

```
_____________________________________________________________________________________________
Layer (type)                        Output Shape      Param #        Connected to
=============================================================================================
input_1 (InputLayer)                [(None, 1, 1)]    0              []

input_2 (InputLayer)                [(None, 1, 1)]    0              []

input_3 (InputLayer)                [(None, 1, 1)]    0              []

concatenate (Concatenate)           (None, 1, 3)      0              ['input_1[0][0]',
                                                                      'input_2[0][0]',
                                                                      'input_3[0][0]']

SharedLayer (LSTM)                  (None, 1, 64)     17408          ['concatenate[1][0]']

GSPC_PRICE_OutLayer (LSTM)          (None, 32)        12416          ['SharedLayer[1][0]']

DJI_PRICE_OutLayer (LSTM)           (None, 32)        12416          ['SharedLayer[1][0]']

NYA_PRICE_OutLayer (LSTM)           (None, 32)        12416          ['SharedLayer[1][0]']

GSPC_PRICE_OutLayer2 (Dense)        (None, 1)         33             ['GSPC_PRICE_OutLayer[1][0]']

DJI_PRICE_OutLayer2 (Dense)         (None, 1)         33             ['DJI_PRICE_OutLayer[1][0]']

NYA_PRICE_OutLayer2 (Dense)         (None, 1)         33             ['NYA_PRICE_OutLayer[1][0]']

=============================================================================================
Total params: 54,755
Trainable params: 54,755
Non-trainable params: 0
```

Figure 151: Multi-Task DL model summary.

```
model.weights[0].shape: TensorShape([3, 256])

model.weights[1].shape: TensorShape([64, 256])

model.weights[2].shape: TensorShape([256])

model.weights[3].shape: TensorShape([64, 128])

model.weights[4].shape: TensorShape([32, 128])

model.weights[5].shape: TensorShape([128])

model.weights[6].shape: TensorShape([64, 128])

model.weights[7].shape: TensorShape([32, 128])

model.weights[8].shape: TensorShape([128])

model.weights[9].shape: TensorShape([64, 128])

model.weights[10].shape: TensorShape([32, 128])

model.weights[11].shape: TensorShape([128])

model.weights[12].shape: TensorShape([32, 1])

model.weights[13].shape: TensorShape([1])

model.weights[14].shape: TensorShape([32, 1])

model.weights[15].shape: TensorShape([1])

model.weights[16].shape: TensorShape([32, 1])

model.weights[17].shape: TensorShape([1])
```

param1 (layer1) = (64 * 4)(64+1) +(64*4) +(64*4) +(64*4) = 17408

param2 (layer2/ putput1) = (32 * 4)(32+64) +(32*4) = 12416

param3 (layer2/ putput2) = (32 * 4)(32+64) +(32*4) = 12416

param4 (layer2/ putput3) = (32 * 4)(32+64) +(32*4) = 12416

param5 (layer2/ putput1) = 32 +1 = 33

param7 (layer2/ putput2) = 32 +1 = 33

param8 (layer2/ putput3) = 32 +1 = 33

Total = 17408 + 12416 + 12416 + 12416 +12416 + 33 + 33 + 33 = 54755

Appendix C

Experimentation Tool for Stock Market Predictions

This appendix contains several essential sections of the code utilized in the development of the Python testing application.

Daily Returns and Volatility Calculations

```
file = "prices.csv"
dataset = None
stocks = '^GSPC, ^DJI, ^IXIC, ^VIX,^NYA'

start = start_date
end = end_date

data = yf.download(stocks, start=start, end = end, interval='1d')
data = data[data['Adj Close', 'BAC'] > 0]

for stock in stocks.split(','):
   returns = 100 * (data.pct_change()[1:]['Adj Close', stock])
   volatility_5d = returns.rolling(5).std()

   column_name = stock.replace('=F', '').replace('^', '')

   if dataset is None:
     dataset = DataFrame(
                 returns.index.values[18:len(returns)],
                 columns=['Date'])

   dataset = dataset.join(DataFrame(
                   data['Adj Close', stock].values[19:],
                   columns=['%s_PRICE' % column_name]))

   dataset = dataset.join(DataFrame(
                   returns.values[18:],
                   columns=['%s_RETURNS' % column_name]))

   dataset = dataset.join(DataFrame(
                   volatility.values[18:],
                    columns=['%s_VOLATILITY'
                                      % column_name]))

   dataset = dataset.join(DataFrame(
                   data['Volume', stock].values[19:],
                   columns=['%s_VOLUME' % column_name]))
```

GARCH Testing

```
function garch(symbol, start_date, end_date):

    stocks = symbol
    start = start_date
    end =  end_date

    # Get daily prices
    prices = yf.download(stocks,
                         start=start,
                         end=end,
                         interval='1d')
    # Calculate Returns
    returns = 100 * (prices.pct_change()[1:]['Adj Close'])

    # Calculate Volatility (Standard Deviation)
    realized_vol = returns.rolling(5).std()

    # 262 business days
    days = 262

    # Getting the start date for start data
    test_plit = returns.iloc[-days:].index

    # Fit GARCH Model
    garch_model = arch_model(returns,
                             mean='zero',
                             vol='GARCH',
                             p=1,
                             q=1).fit(disp='off')

    # Forecast Volatility
    forecast = garch_model.forecast(start=test_plit[0])

    # MSE calculation
    mse_garch = np.sqrt(
                  mse(
                      realized_vol[-days:] / 100,
                      np.sqrt(
                      forecast.variance.iloc[-len(days):]/100
                             )
                     )
                    )

    # Return the MSE value
    return mse_garch
```

ARIMA Testing

```
function arima(symbol, start_date, end_date):


    stocks = symbol
    start = start_date
    end =  end_date


    # Get daily prices
    prices = yf.download(stocks,
                         start=start,
                         end=end,
                         interval='1d')
    # Calculate Returns
    returns = 100 * (prices.pct_change()[1:]['Adj Close'])

    # Calculate Volatility (Standard Deviation)
    realized_vol = returns.rolling(5).std()

    # 262 business days
    days = 262

    # Initialize an empty list for predictions
    predictions = []

    # Loop through the test data
    for t in range(len(test_data)):

        # Create an ARIMA model
        model = ARIMA(history, order=(p=p, o=o, q=q), trend='n')
        model_fit = model.fit()

        # Make a forecast
        forecast = model_fit.forecast()

        # Append the forecast to the predictions list
        predictions.append(forecast[0])

        # Update the history with the current test data point
        history.append(test_data[t])

    # Calculate the scaled test MSE
    mse_arima = mse(
                      realized_vol.shift(1)[-n:]/100,
                      predictions /100)

    # Return the ARIMA MSE
    return mse_arima
```

Appendix D
Correlation Matrices

Table 1: Correlation matrix for symbols used in multivariate testing (Chapter 5).

Bull Market

	BAC	JPM	C	WFC	MS	GS	DB	IBM	WMT	GOLD	OIL
BAC	1	-0.12	0.72	-0.12	0.66	0.13	0.73	-0.58	-0.43	-0.53	-0.12
JPM	-0.12	1	-0.52	0.92	0.15	0.84	-0.39	0.69	0.81	0.53	0.15
C	0.72	-0.52	1	-0.6	0.67	-0.44	0.62	-0.8	-0.7	-0.88	-0.51
WFC	-0.12	0.92	-0.6	1	-0.03	0.82	-0.41	0.74	0.86	0.63	0.27
MS	0.66	0.15	0.67	-0.03	1	0.28	0.58	-0.38	-0.25	-0.55	-0.31
GS	0.13	0.84	-0.44	0.82	0.28	1	0.05	0.55	0.57	0.53	0.41
DB	0.73	-0.39	0.62	-0.41	0.58	0.05	1	-0.52	-0.64	-0.39	0.2
IBM	-0.58	0.69	-0.8	0.74	-0.38	0.55	-0.52	1	0.86	0.89	0.53
WMT	-0.43	0.81	-0.7	0.86	-0.25	0.57	-0.64	0.86	1	0.68	0.25
GOLD	-0.53	0.53	-0.88	0.63	-0.55	0.53	-0.39	0.89	0.68	1	0.69
OIL	-0.12	0.15	-0.51	0.27	-0.31	0.41	0.2	0.53	0.25	0.69	1

Bear Market

	BAC	JPM	C	WFC	MS	GS	DB	IBM	WMT	GOLD	OIL
BAC	1	0.53	0.39	0.87	0.38	0.72	0.78	-0.09	-0.29	0.48	0.56
JPM	0.53	1	0.26	0.65	0.74	0.82	0.85	0.55	-0.09	0.53	0.58
C	0.39	0.26	1	0.06	0.65	0.12	0.35	-0.14	-0.39	-0.49	-0.31
WFC	0.87	0.65	0.06	1	0.29	0.8	0.78	0.07	-0.17	0.76	0.74
MS	0.38	0.74	0.65	0.29	1	0.64	0.73	0.43	-0.32	0.04	0.18
GS	0.72	0.82	0.12	0.8	0.64	1	0.94	0.45	-0.2	0.73	0.76
DB	0.78	0.85	0.35	0.78	0.73	0.94	1	0.36	-0.29	0.57	0.67
IBM	-0.09	0.55	-0.14	0.07	0.43	0.45	0.36	1	0.31	0.34	0.41
WMT	-0.29	-0.09	-0.39	-0.17	-0.32	-0.2	-0.29	0.31	1	0.07	0.03
GOLD	0.48	0.53	-0.49	0.76	0.04	0.73	0.57	0.34	0.07	1	0.87
OIL	0.56	0.58	-0.31	0.74	0.18	0.76	0.67	0.41	0.03	0.87	1

Table 2: Correlation matrix for symbols used in multitasking testing (Chapter 6).

Bull Market	GSPC_PRICE	DJI_PRICE	GSPC_RETURNS	DJI_RETURNS	GSPC_VOLATILITY	DJI_VOLATILITY	GSPC_VOLUME	DJI_VOLUME
GSPC_PRICE	1	0.99	0.1	0.09	-0.54	-0.55	-0.28	-0.46
DJI_PRICE	0.99	1	0.11	0.11	-0.57	-0.58	-0.26	-0.45
GSPC_RETURNS	0.1	0.11	1	0.97	0.06	0.06	-0.14	-0.2
DJI_RETURNS	0.09	0.11	0.97	1	0.06	0.06	-0.14	-0.18
GSPC_VOLATILITY	-0.54	-0.57	0.06	0.06	1	0.97	0.26	0.36
DJI_VOLATILITY	-0.55	-0.58	0.06	0.06	0.97	1	0.26	0.35
GSPC_VOLUME	-0.28	-0.26	-0.14	-0.14	0.26	0.26	1	0.87
DJI_VOLUME	-0.46	-0.45	-0.2	-0.18	0.36	0.35	0.87	1
Bear Market								
	GSPC_PRICE	DJI_PRICE	GSPC_RETURNS	DJI_RETURNS	GSPC_VOLATILITY	DJI_VOLATILITY	GSPC_VOLUME	DJI_VOLUME
GSPC_PRICE	1	1	0.08	0.08	-0.68	-0.66	-0.62	-0.33
DJI_PRICE	1	1	0.09	0.09	-0.67	-0.66	-0.64	-0.33
GSPC_RETURNS	0.08	0.09	1	0.99	0.05	0.05	-0.02	0.01
DJI_RETURNS	0.08	0.09	0.99	1	0.06	0.07	-0.01	0.01
GSPC_VOLATILITY	-0.68	-0.67	0.05	0.06	1	0.99	0.62	0.44
DJI_VOLATILITY	-0.66	-0.66	0.05	0.07	0.99	1	0.61	0.42
GSPC_VOLUME	-0.62	-0.64	-0.02	-0.01	0.62	0.61	1	0.73
DJI_VOLUME	-0.33	-0.33	0.01	0.01	0.44	0.42	0.73	1
Volatile Market								
	GSPC_PRICE	DJI_PRICE	GSPC_RETURNS	DJI_RETURNS	GSPC_VOLATILITY	DJI_VOLATILITY	GSPC_VOLUME	DJI_VOLUME
GSPC_PRICE	1	0.92	0.08	0.08	-0.57	-0.59	-0.27	-0.22
DJI_PRICE	0.92	1	0.09	0.09	-0.71	-0.72	-0.46	-0.45
GSPC_RETURNS	0.08	0.09	1	0.98	-0.06	-0.04	-0.1	-0.14
DJI_RETURNS	0.08	0.09	0.98	1	-0.06	-0.04	-0.08	-0.12
GSPC_VOLATILITY	-0.57	-0.71	-0.06	-0.06	1	0.99	0.64	0.71
DJI_VOLATILITY	-0.59	-0.72	-0.04	-0.04	0.99	1	0.65	0.7
GSPC_VOLUME	-0.27	-0.46	-0.1	-0.08	0.64	0.65	1	0.87
DJI_VOLUME	-0.22	-0.45	-0.14	-0.12	0.71	0.7	0.87	1

Appendix E

Deep Learning: Dynamic Weights Optimization

In this experiment, our objective was to enhance the performance of a two-task model by dynamically adjusting the weights assigned to its tasks during the training process. To accomplish this, modifications were introduced to the built-in method, specifically the **on_epoch_end()** function. This customization enabled us to read and update the weights, often referred to as "K values" in the context of Keras, in real-time without the need to recompile the entire model.

```
class MultiTaskMultiInputModel(BaseModel):
    """
    Multi task model
    """
    code_version = "MT_V2"
    file_name = 'test'

    alpha = K.variable(0.5)
    beta = K.variable(0.5)

    class MyCallback(tf.keras.callbacks.Callback):
        def __init__(self, alpha, beta):
            self.alpha = alpha
            self.beta = beta

        # customize your behavior
        def on_epoch_end(self, epoch, logs={}):
            self.alpha = self.alpha - 0.01
            self.beta = self.beta + 0.01
            K.get_value(self.alpha)
            K.get_value(self.beta)

            losses = np.array([v for k, v in logs.items() if
                                   k in [
                                        'val_GSPC_PRICE_OutLayer2_loss',
                                        'val_DJI_PRICE_OutLayer2_loss',
                                        'GSPC_PRICE_OutLayer2_loss',
                                        'DJI_PRICE_OutLayer2_loss']],
                                 dtype=np.float64)

            losses = (losses - 0.5 * losses.min()) /
                     (losses.max() - 0.5 * losses.min())
            losses = losses / np.sum(losses)

            K.set_value(self.alpha, losses[0])
            K.set_value(self.beta, losses[1]
```

This approach represents a dynamic and adaptive way to fine-tune the model behaviour as it learns from the data. By adjusting the weights interactively during training, the aim is to optimize the model gradient descent behaviour and, ultimately, improve its overall performance. Such dynamic weight adjustments can be especially valuable when dealing with complex tasks or evolving data patterns, as they enable the model to respond and adapt more effectively to changing conditions.

This experiment reflects the versatility and flexibility of deep learning frameworks such as Keras, which enable researchers and practitioners to tailor models to specific needs and examine innovative strategies for improving model training and performance.

During the model training process, the weights were continuously updated for each epoch using a callback mechanism. Specifically, two weights were involved in this process: alpha, which corresponds to the weight assigned to the first task, and beta, which represents the weight for the second task. These weight updates occurred systematically throughout the training, could influence the model performance and learning behaviour as it progressed through epochs.

```
if loss_callback and len(predict_cols) > 1:
    self.alpha = K.variable(weights[0])
    self.beta = K.variable(weights[1])
    call_backs.append(self.MyCallback(self.alpha, self.beta))
    loss_weights = [self.alpha, self.beta]
```

It is important to note that this test was done to experiment with dynamic weights and use this as the seed for the next research. The method used to update the weights is not based on any statistical model and it was more about understanding the behaviour and the impact of such change on the model performance. Therefore, due to lack of stability, this test was taken out from thesis and replaced with more stable one detailed in Section 7.2.3.4: Fine-Tuning the Loss Function: Determining Optimal Weights.