5-2024

EFFECTIVENESS OF CNN-LSTM MODELS USED FOR APPLE STOCK FORECASTING

Ethan White

EFFECTIVENESS OF CNN-LSTM MODELS USED FOR APPLE STOCK FORECASTING

A Project

Presented to the

Faculty of

California State University,

San Bernardino

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

in

Information Systems & Technology:

Business Intelligence and

Data Analytics

---

by

Ethan White

May 2024

EFFECTIVENESS OF CNN-LSTM MODELS USED FOR APPLE STOCK FORECASTING

A Project

Presented to the

Faculty of

California State University,

San Bernardino

---

by

Ethan White

May 2024

Approved by:

Dr. Conrad Shayo, Committee Member, Chair

Dr. Sepideh Alavi, Committee Member, Reader

Dr. Conrad Shayo, Chair, Information and Decision Sciences Department

ABSTRACT

This culminating experience project investigates the effectiveness of convolutional neural networks mixed with long short-term memory (CNN-LSTM) models, and an ensemble method, extreme gradient boosting (XGBoost), in predicting closing stock prices. This quantitative analysis utilizes recent AAPL stock data from the NASDAQ index. The chosen research questions (RQs) are: RQ1. What are the optimal hyperparameters for CNN-LSTM models in stock price forecasting? RQ2. What is the best architecture for CNN-LSTM models in this context? RQ3. How can ensemble techniques like XGBoost effectively enhance the predictions of CNN-LSTM models for stock price forecasting?

The research questions were answered through a thorough quantitative analysis involving data preprocessing, feature engineering, and model evaluation, using various Python scripts designed for this analysis. The findings are: RQ1. reveals that adjusting hyperparameters, such as learning rates and epochs, significantly improves model performance; RQ2. deemed a multi-layered CNN-LSTM structure with attention mechanisms as the most effective for this use case; and RQ3. showed that XGBoost as an ensemble method did not work as planned, indicating a much more complex interplay between ensemble methods and neural network models. The conclusions are: RQ1. adjusting hyperparameters, such as learning rates and epochs, improves the performance of CNN-LSTM models. RQ2. multi-layered CNN-LSTM architectures with

attention mechanisms are the most effective architecture for predicting stock prices. RQ3. ensemble methods like XGBoost, when combined with CNN-LSTM models, did not improve prediction accuracy as expected, suggesting a complex interplay between these techniques. Areas for further study include the automation of hyperparameter tuning techniques such as GridSearch and Bayesian optimization, further exploration of the integration of ensemble methods with neural network models, and the application of CNN-LSTM architectures to other forms of financial data beyond closing stock prices.

ACKNOWLEDGEMENTS

This culminating experience project on the effectiveness of CNN-LSTM models in stock forecasting has been a significant milestone in my academic journey. I owe my deepest gratitude to Dr. Shayo, who has guided me with his supply chain expertise. His active support in navigating the complexities of this research has been appreciated. I am equally grateful for the support of Dr. Sepideh, my committee member, whose experience in the supply chain has also enhanced this quantitative analysis. I am truly fortunate to have had such distinguished scholars on my committee.

I would also like to extend my heartfelt thanks to my current employer and coworkers for their patience and support as I continued to pursue my education at California State University, San Bernardino (CSUSB). Special acknowledgment goes to my supervisor, Chenkuan Tiow, whose understanding and flexibility were instrumental in balancing my professional responsibilities with my academic endeavors.

Furthermore, I am immensely grateful for the CSUSB Business Job Fairs held every spring. This event made my current employment possible, opening the door to new opportunities that have enriched my professional and personal life. The university's commitment to fostering career development and employment opportunities has had a lasting impact on my career trajectory, for which I am forever thankful.

DEDICATION

I dedicate this analysis to my real-life superhero and forever role model, my mom. I am so proud of your willingness to go back to college during my last few years of high school. Now look at us, we are both graduating with our master's degrees at the same time and from the same school. Who would have thought? I am forever indebted to your stick-to-it-veness and sacrifices made to be able to achieve such a rare feat. I am so proud and honored to call myself your son.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER ONE:

INTRODUCTION

The stock market remains as a pivotal institution where capital flows, investments are made, and the economy is impacted. Within it, billions of dollars change hands every day. Consequently, automating accurate stock price predictions has become a primary goal for supply chain researchers and investors globally (Zhu et al., 2023). Innovations such as artificial intelligence (AI) models are effective at recognizing patterns, and they could be the key to improving stock trend forecasting for financial researchers. Before covering the specifics of AI architectures, it is crucial to have a deeper understanding of neural networks and their current function for automation.

Background Information

According to research by Islam and team, a neural network is a form of artificial intelligence that uses algorithms to imitate the structure of the human brain (Islam et al., 2019). Neurons in the human brain receive electrical or chemical stimuli through dendrites and transfer output signals via axons. Axons establish connections with other neurons at junctions known as synapses, delivering their output signals to other neurons in an endless cycle of transmission. Artificial neural networks consist of interconnected units, often known as neurons. The link between the network neurons in the hidden layer functions similarly to the synapses in the human brain. The neurons establish

connections between processing elements, and the arrangement and weights of these connections affect the output. Unlike solely digital models that manipulate binary code, neural networks operate with more complex computations. This form of artificial intelligence excels when given a substantial amount of previous example data for training (Islam et al., 2019).

Architectures

Existing Architectures

Within the realm of artificial intelligence, neural network architectures are essential for solving complex predictive problems. To provide a structured overview of the primary neural network architectures that have influenced recent machine learning advancements, Table 1 below summarizes eight influential models. Each architecture is evaluated based on its design, advantages, and where it is suitable for closing stock price analysis.

Table 1 - Eight Influential Models

Model	Description	Advantages	Sustainability for Project
Multilayer Perceptron (MLP)	A forward-feed neural network with multiple layers.	Can approximate any function.	Not selected due to lower performance capturing time-series data.

Model	Description	Advantages	Sustainability for Project
Convolution Neural Network (CNN)	Uses convolutional layers, and fully connected layers.	Excellent for spatial data like imagery.	Selected because it can handle time-series data, treating it as a sequence of patterns
Recurrent Neural Network (RNN)	Processes sequences by iterating through elements.	Good for sequence predictions.	Not selected due to difficulty to train and issues with holding long-term dependencies.
Long Short-Term Memory (LSTM)	Type of RNN that can learn long-term dependencies.	Prevents the vanishing gradient problem.	Selected for its strength in handling long sequences, crucial for stock price forecasting.
Feed-Forward Neural Networks	Simplest type of ANN, connections do not form cycles.	Simple and fast to train.	Not selected because it cannot handle sequences, crucial for time series analysis.
Generative Adversarial Networks (GAN)	Consists of two networks, competing against each other.	Good for generating new data.	Not selected as it is better suited for data generation, not data prediction.
Residual Networks (ResNet)	Utilizes skip connections to jump over some layers.	Efficient for identifying gradient issues in deep networks.	Not selected due to its complexity and inefficiency in time series prediction compared to both CNN and LSTM

Model	Description	Advantages	Sustainability for Project
Transformers	Based on self-attention mechanisms instead of sequence aligned RNNs or convolution	Highly parallelizable and effective in handling long-range dependencies.	Not selected because it is complex and resource-intensive, making it suitable for larger datasets and natural language processing.

After presenting the advantages and applications of architectures detailed within works such as Goodfellow et al. (2014); Haykin (1999); He, Zhang, Ren, & Sun (2016); Hochreiter & Schmidhuber (1997); LeCun, Bengio, & Hinton (2015); O'Shea & Nash (2015); Rosenblatt (1958); and Vaswani et al. (2017), the rationale for selecting specific models depends on their ability to perform time series forecasting. Thus, CNN and LSTM networks were chosen based on their ability to process sequential data effectively.

According to O'Shea and Nash, the CNN model resembles feed-forward neural network architectures, where the neurons possess adjustable weights and biases (O'Shea & Nash, 2015). This neural network architecture can be divided into four major parts: the input layer, the convolutional layer, the pooling layer, and the fully connected layers. Convolution is often used for signal and image recognition. However, it has recently adapted to handle time series prediction applications by utilizing 2D mapping. Time series forecasting uses historical time-stamped data to create scientific forecasts. The process entails constructing

models and using them to inform future strategic decision-making. The forecasting process is unique because, during the process, the result of the models is unknown, and it can only be found through the meticulous examination of the data (Tableau, 2003-2024). When applying CNN to time series inputs, the data would be treated as an image-like structure, where the horizontal axis represents the temporal dimension, and any other dimensions are represented by the other axes. After mapping the dimensions, patterns within the data can be identified accordingly (O'Shea & Nash, 2015).

The RNN model also shares similarities with feed-forward neural network architectures since the initial layer is computed by multiplying the sum of weights and features. After the calculation, the neurons store a small amount of information throughout each algorithm stage, which is then utilized for error correction during the backpropagation process. A basic RNN block comprises five steps: block input, input gate, forget gate, cell, and block output (Islam et al., 2019). The LSTM model, pioneered by Hochreiter & Schmidhuber in 1997, addressed the challenge of learning long-term dependencies, a way for a computer to detect cohesion between phrases. The learning capacity of LSTM has had a significant impact in terms of practical applications and theoretical advancements, leading to its recognition as an innovative learning model (Van Houdt, Mosquera, & Nápoles, 2020).

Accurately predicting stock market movements has attracted significant attention due to its unpredictable patterns. The growing popularity of purchasing

stocks can be attributed to convenient information accessibility, online brokerage programs, the dynamic market, and the pursuit of financial literacy overall. Past research has shown that, when combined, CNN-LSTM models are resilient to noisy input data and can differentiate between meaningful information and irrelevant disturbances. These models are known for generating accurate predictions, and they can efficiently combine multiple types of data sources and identify relevant characteristics for forecasting (Zhu et al., 2023). Further research is necessary to ensure the accuracy of machine learning models to detect outliers when applied to fluctuating stock price data (Zhu et al., 2023). The research questions chosen are addressed within the research scope.

Problem Statement

The main objective of this culminating experience project is to analyze the effectiveness of neural network models used for stock price forecasting. Previous research (Zhu et al., 2023), and most references present at the end of this analysis, show that there is a need to study the forecasting effectiveness of CNN and LSTM models. We will analyze these models' effectiveness while providing explanations for RQs relating to the topic. This analysis of convolutional neural network models draws inspiration from Yuzhun Liang's area for further study, and the incorporation of long short-term memory is motivated by the recent work of Jonathan Cahyadi and Amalia Zahra, who applied CNN-LSTM to predict bitcoin prices (Liang, 2019; Cahyadi & Zahra, 2024).

Research Scope

This research's main goal is to analyze the effectiveness of CNN LSTM models when combined to forecast time series data, more specifically the forecasting of closing stock prices. A detailed exploration of the model's performance and adaptability to the dynamic nature of financial markets forms the core of this research. The following are research questions suggested by prior research that are analyzed within this project:

RQ1: What are the optimal hyperparameters for CNN-LSTM models in stock price forecasting, and how can they be updated to positively impact the model's performance and ability to generalize? (Zhu et al., 2023)

RQ2: What is the best architecture of CNN-LSTM models being used for stock price forecasting regarding the number of convolutional layers, LSTM layers, batch sizes/maximum epochs, and python libraries used? (Zhu et al., 2023)

RQ3: How can ensemble techniques, such as extreme gradient boosting, effectively combine the predictions of multiple CNN-LSTM models for stock price forecasting? (Zhu et al., 2023)

This culminating experience project is organized as follows: Chapter 2 reviews the literature focusing on where the research questions came from. Chapter 3 provides the methods used to answer the research questions, and

Chapter 4 implements the research methods provided in Chapter 3 by covering the data collection, analysis and providing the research findings. Chapter 5 provides a discussion of the findings, conclusions, and areas for further study.

CHAPTER TWO:

LITERATURE REVIEW

Understanding the volatility of closing stock prices has evolved since the market's inception. Accurately predicting potential shifts is essential for making educated investment decisions and utilizing technology can make this process more streamlined. The past research included within this research shows that neural network models have higher accuracy than linear statistical-based methods when applied to non-linear data. More specifically, hybrid non-linear models have generated encouraging results suggesting enhanced forecast accuracy during periods of market volatility, like the COVID-19 pandemic. Nevertheless, the implementation of neural networks can be improved, which will be examined further within this analysis. Researchers are now investigating the most efficient optimization techniques in this developing field, including optimal hyperparameters and the integration of neural network models as ensemble models. This literature review includes the comparative behavior of international stock indices, groundbreaking neural network research in financial forecasting, and the evolution of neural networks used as predictive methodologies.

Comparative Behavior of International Stock Indices

As defined previously, external variables impact financial forecasting. However, regional differences also play a critical role in shaping the behaviors of global stock markets. A well-known comparative study, "Industrial Structure and

the Comparative Behavior of International Stock Market Indices," published in *The Journal of Finance* in 1992, documents the behaviors. The analysis reveals differences in volatility levels among national stock markets, which are present after accounting for nominal and inflation differences through currency conversions. The research found that stock markets in places like Canada and the Netherlands do not change as much, which means they have low volatility, while markets in Hong Kong and South Africa change a lot more, showing high volatility (Roll, 1992, p. 37). It was found that changes in the value of a country's currency play a role in the behavior of national stock market indices, but this influence is smaller than the effect of the country's industrial structure. Exchange rates significantly affect stock market movements, but how much varies by country. The results reveal the intricate complexity of international stock market indices, offering helpful information for stakeholders operating in global financial markets (Roll, 1992). Roll's research emphasizes external factors affecting stock price forecasts, making it a challenging endeavor, and underscores the role that location plays in generating predictions.

Groundbreaking Research

Seminal works—Kimoto et al., 1990, Kamijo & Tanigawa, 1990, and (Ahmadi, 1990)— have applied ANN architectures to forecast foreign indices such as the Tokyo Stock Exchange Prices Index (TOPIX) by closely monitoring each stock index's reversal patterns. Contemporary research has focused on

applying AI techniques, particularly ANN, to predict recorded stock price trends. Studies such as (Yoon & Swales, 1991), (Choi, Lee, & Rhee, 1995), and (Trippi & DeSieno, 1992) were some of the first to use ANN models to predict stock index futures, namely the S&P 500. Additionally, (Duke & Long, 1993) extended this approach to German government bond futures. These early investigations primarily focused on ANN's application in stock market prediction.

As research progressed, scholars (Hiemstra, 1995) introduced hybrid models, integrating fuzzy expert systems with ANN to capture the complexities of market dynamics (Tsaih, Hsu, & Lai, 1998). These researchers further advanced this trend by combining rule-based techniques with ANN to forecast the S&P 500 index futures' daily direction of change. Moreover, researchers like (Kohara, Ishikawa, Fukuhara, & Nakamura, 1997) began incorporating prior knowledge to enhance prediction performance. However, challenges arose due to the noisy and non-stationary nature of stock market data, as noted by (Lawrence, Tsoi, & Giles, 1996), leading to issues like overfitting and local convergence of gradient descent algorithms commonly used to train ANNs. This was also addressed in the research of (Abu-Mostafa & Atiya, 1996) where the "learning from hints" algorithm was created to identify familiar market information and uses it to improve its predictive capabilities. The algorithm adapts by fine-tuning itself according to its alignment with the clues and the information provided and it does not rely on the knowledge of each scenario result, which makes it valuable even with restricted data. However, recent advancements propose a novel hybrid

approach that combines genetic algorithms, also known as GAs, with ANNs to mitigate data complexity (Kim & Han, 2000). This hybrid model not only optimizes connection weights but also determines optimal thresholds for feature discretization, thus enhancing classifier generalizability by reducing dimensionality. The concepts applied within this early research underscore the potential to explore broader algorithmic optimization strategies, aligning with the evolving landscape of financial forecasting techniques.

2.1 – RQ1: What are the optimal hyperparameters for CNN-LSTM models in stock price forecasting, and how can they be updated to positively impact the model's performance and ability to generalize?

When researching optimal hyperparameters for CNN-LSTM applications, we found research that could be adapted to the optimization of CNN-LSTM model training. According to Hanifi et al.'s 2024 study, hyperparameters are external parameters whose values are not directly learned by ANN or RNN models as they are defined outside of training to optimize the learning of the chosen models (Hanifi et al., 2024, p. 2). In this comparative study, hyperparameters such as the number of neurons, batch size, epochs number, and the activation functions were optimized using three optimization strategies. The epochs number refers to the number of times that the model has seen the entire dataset, and epochs are split up into smaller divisions for processing called batch sizes (Hanifi et al., 2024, p. 2). We learned from Hanifi's research that

selecting efficient hyperparameter values is critical for avoiding overfitting and other discrepancies. Hyperparameter selection varies based on the domain and should be optimized for each dataset through a process known as hyperparameter tuning. However, determining how many hyperparameter combinations to test is critical, as it affects the computational cost of the model being used. Hanifi's research explores automatic search algorithms, such as grid search and random search, to help overcome the challenges of manual hyperparameter estimation, especially when applied to neural networks (Hanifi et al., 2024).

Some approaches to optimize hyperparameters included within Hanifi's research were grid search and sequential model-based optimization (SMBO). Grid search explores hyperparameter combinations but suffers due to the high computation costs of neural networks models (Wu et al., 2019). Another source introduces a new grid search method GridsearchWEF, which reduces the time needed to find the best settings for machine learning models (Zhao et al., 2024, p. 111362) but was not applied with CNN-LSTM. Highlighting the inefficacies of random search methods, Bergstra et al. (2011) prompted a search for alternatives that balance efficacy and computational efficiency. Enter sequential SMBO, which is an optimization strategy that uses past training data to iteratively refine hyperparameters. An example of this strategy is seen through the work of Masum et al. (2021), highlighting Bayesian optimization for the detection of network intrusions. Another example of this strategy was research by Zhu et al.

(2022) exploiting a Tree-structured Parzen Estimator (TPE) to fine-tune wind power prediction models. However, it was Hanifi et al. (2024) who compared Scikit-opt, Hyperopt, and Optuna python-language libraries used for optimizing CNN and LSTM models, which can be applied to existing stock forecasting models. Hanifi's meticulous analysis not only advocates for SMBO as a viable alternative to grid or random searches but also underscores the significance of addressing randomness to increase model accuracy.

This section sheds light on hyperparameter optimization and sets the stage to discuss the literature containing the *best* architecture to use for this data analysis project.

2.2 – RQ2: What is the best architecture of CNN-LSTM being used for stock price forecasting regarding the features generated, hyperparameters used, and modifications made to the model itself?

When searching for the *best* architecture, we found that the approach of CNN-LSTM architecture varied between past research studies, in terms of the number of layers, filter sizes, and hyperparameters for each model. After reviewing over 5 different approaches to the CNN-LSTM model found in modern research, we noticed that these models used at least two convolutional layers and filter sizes ranging from 32 to 64 (Livieris et al., 2021; Staffini, 2022; Alkhatib et al., 2022; Yang & Chang, 2020; Song & Choi, 2023). This is all dependent on the size of the dataset, as this will determine how many epochs are generated by

the model. Some modern forecasting research creates innovative new features used in their forecasting. Research done by Song & Choi of 2023 uses historical stock price data from DAX, Dow Jones, and the S&P 500 to forecast the one-time-step and multiple-time-step closing prices of these indices through the integration of neural network models such as CNNs, LSTMs, and gated recurrent units (GRUs). Several of the features used by the models for forecasting include the daily stock prices for each index, trading volume, change in stock prices, and a novel feature called the medium. The medium was an average of the stock prices' highs and lows that the models use to make better predictions. Through calculating the medium, the forecasting model can focus on a more stable value that represents the trends of an index rather than being swayed by its volatility (Song & Choi, 2023, p. 13). Other modern research makes modifications to the neural network model.

Eapen et al. (2019) conducted further research that introduces a new deep learning model combining CNN with bi-directional LSTM units. This model aims to improve the accuracy of predicting stock market indices, with a focus on the S&P 500 index. These researchers look at what happens when they change various parts of the models, analyzing what happens when they change the number of bidirectional LSTM units and the size of the CNN kernels. The team used six python packages to facilitate their display of stock price forecasts. In summary, the study conducted by Eapen, and team emphasizes the effectiveness of a hybrid neural network architecture consisting of CNN for

recognizing stock price trends and a LSTM network for retaining the temporal sequence of events. Making modifications to a model's framework, in terms of their layering and filtering, could impact their effectiveness in the future.

The above research contains effective architectures for stock forecasting. However, there is still a gap that needs to be addressed. An architecture that applies ensemble techniques to CNN-LSTM models should be considered for this research.

2.3 – RQ3: How can ensemble techniques, such as extreme gradient boosting, effectively combine the predictions of multiple CNN-LSTM models for stock price forecasting?

When searching for research on CNN-LSTM models incorporating ensemble techniques for error processing, we discovered the research of Zhu et. al, 2023. This research displayed a CNN-LSTM model, the CNN enhanced by attention mechanisms, using a decision-tree algorithm to improve its ability to generalize called XGBoost. It does this by focusing on the errors of the predictions, and uses the information collected from the trees to make predictions closer actuals, in this case actual stock prices (Zhu et al., 2023, p. 362). This ensemble methods' guesses can be imperfect, and these imperfections or errors are known as residuals. When coupled with other predictive techniques, the XGBoost algorithm employs subsequent decision trees to identify these imperfections and fine-tune the learning model. This research shows that the

combination of XGBoost with CNN-LSTM offers better insights and higher accuracy for predicting time series data.

The combination of stand-alone neural network models into hybrids, such as CNN-LSTM, opens avenues for ensemble modeling approaches. Analogous to a council of experts using collected data to pool insights for decision making, ensemble models aggregate predictions from all models as sources. Comparative research by Song & Choi (2023) pitted hybrid and ensemble models against traditional statistical methods in stock price prediction. In one-time-step forecasting, the latest models eclipsed their traditional counterparts in over 48% of cases, while ensemble models outperformed traditional methods in more than 81% of multi-time-step forecasting instances. These methodologies show a shift towards using more nuanced predictive analytics within financial forecasting, such as boosting (Song & Choi, 2023, p. 1).

Precise predictions improve investor decision-making, and better investments benefit the stock market. Having reviewed the existing literature on the application of modern CNN-LSTM models in stock forecasting, we applied what we learned to adapt the model created in Zhu et al., 2023 fitted to closing price data from the APPL ticket of the NASDAQ index. The Zhu research will be our primary literature for this analysis, and we will adapt their methodologies to better understand the techniques used in modern research and identify potential areas for improvement.

CHAPTER THREE:

RESEARCH METHODS

The methodologies from Zhu et al.'s research, 2023, are the primary literature for this analysis and are used as secondary, quantitative data to answer the three RQs. Zhu's research includes the discussion of hyperparameters, the implementation of ensemble methods, and their research results show that this is one of the best models used for stock prediction. Fortunately, prior research has already implemented the architecture provided in the primary literature, authored by Zhuangwei Shi, Yang Hu, Guangliang Mo, and Jian Wu on their GitHub repository, and it has been adjusted to fit the criteria (Shi et al., 2022). The scripts are titled: arima_APPL.py, lstm_APPL.py, xgboost_APPL.py, model_APPL.py, main_APPL.py, and lastly, utils_APPL.py. This methodology utilizes various Python libraries, notably Keras for the neural network models and XGBoost for ensemble learning. Data handling and preprocessing tasks are performed on the chosen 5-year APPL dataset using pandas and scikit-learn, ensuring that the original data is properly scaled and cleaned before processing. The adjusted Python scripts perform an evaluation of the methods used and they generate a month's worth of predictions for closing prices on the APPL ticket of the NASDAQ index. Understanding the methodology will serve as a foundation for discussing the findings as well as potential improvements that could be made within time series forecasting.

3.1 – RQ1: What are the optimal hyperparameters for CNN-LSTM models in stock price forecasting, and how can they be updated to positively impact the model's performance and ability to generalize?

Chapter 2 introduced common benchmark hyperparameters seen through the literature for testing modern CNN-LSTM models, which include maximum epochs of 50 and batch sizes of 32. Additionally, typical testing parameters such as lookback periods, dropout rates, and validation splits of 0.1 were considered. These parameters served as a baseline for identifying the optimal hyperparameters of this analysis, which were manually selected based on their effectiveness in preprocessing tests and generated figures. To answer this RQ, we focus on the optimal hyperparameters chosen for the CNN and LSTM models, configured to predict closing prices and analyze trends within the AAPL dataset.

We chose to adopt the hyperparameters used in the primary literature conducted by Zhu et al. in 2023 as these are the most optimal for our dataset. GridSearch investigated the potential integration of systematic hyperparameter optimization within our models, although it was not implemented due to logic issues with this process. For the LSTM model, the optimal configuration included utilizing sequences of 10 historical data points, each corresponding to the stock prices of previous trading days, to predict the subsequent value. This setup effectively encapsulates two weeks of stock market activity in each input sequence for forecasting. The model was trained over 50 epochs to maximize

training without overfitting, and it used a batch size of 32 to balance computational efficiency by processing information in smaller segments. The LSTM layers were configured with 50 units each, which optimizes the model's capability to learn from temporal patterns in the closing prices. A learning rate of 0.01 for the Adam optimizer was selected to ensure efficient convergence during the model's training process. The attention-based CNN model was optimized with a look-back period of 60 days, recognizing the importance of this duration in capturing relevant stock price trends. The model incorporated 64 convolutional filters, a kernel size of 3, and a dropout rate of 0.5, which was selected to optimize feature extraction. The learning rate for this model was set at 0.001, aimed at achieving steady progress in the model's training process.

Once the hyperparameters were chosen, they were then implemented in the model scripts: lstm_APPL.py and model_APPL.py. The hyperparameter selection process was aimed at maximizing learning and generalization while preventing overfitting. The manual tuning of hyperparameters, as opposed to using GridSearch, was dictated by the programmatical challenges faced when implementing the latter method. Therefore, the hyperparameter configurations for the CNN and LSTM models were carefully customized to enhance their forecasting performance.

3.2 – RQ2: What is the best architecture of CNN-LSTM being used for stock price forecasting regarding the features generated, hyperparameters used, and modifications made to the model itself?

The best architecture for CNN-LSTM in stock price forecasting, identified in our analysis, integrates the attention mechanism within the CNN layers and the LSTM components to effectively analyze time-series data. The ACNN-LSTM model, originally implemented by Zhu et al. (2023), uses attention-based mechanisms in the CNN to prioritize data points impacting future stock price predictions or feature selection. This model is implemented in the model_APPL.py script, and it demonstrates how attention layers can improve feature selection and extraction.

In the lstm_APPL.py script, the LSTM is implemented with a configuration that captures temporal dependencies and patterns over varying time intervals. This includes declaring the most optimal hyperparameter values for the time steps/lookback period, batch size, and number of epochs. These were each optimized to balance learning efficiency of each model, keeping overfitting in mind. Further enhancements to the model's architecture include the integration of XGBoost, as shown in the xgboost_APPL.py script. This addition aims to refine the model's predictions by addressing residuals and incorrect forecasts.

In conclusion, the best architecture for CNN-LSTM in stock price forecasting within this study involves a nuanced combination of ACNN with LSTM, optimized through strategic hyperparameter tuning and further enhanced

with XGBoost. This configuration maximizes the model's ability to learn from historical data and make accurate predictions on the APPL dataset.

3.3 – RQ3: How can ensemble techniques, such as extreme gradient boosting, effectively combine the predictions of multiple CNN-LSTM models for stock price forecasting?

This research question explores the application of XGBoost as an ensemble method to enhance the predictions made by CNN-LSTM models adjusted. The XGBoost technique used can be outlined by the following phases: In the first phase, the stand-alone CNN-LSTM models are trained on segments of the NASDAQ stock dataset. The CNN model focuses on short-term price movements using its convolutional layers, while the LSTM model captures longer-term dependencies with its LSTM layers. Once trained, the output from these models is combined in a new dataset. This combination aligns the predicted closing prices with the actual closing prices from the original APPL dataset. The XGBoost ensemble method is applied to the aggregated predictions dataset in the next phase. The use of XGBoost's gradient boosting capabilities is for the correcting of errors present in the initial predictions. The success of this approach is directly correlated to the hyperparameter tuning of the XGBoost method, and parameters such as learning rate and tree complexity were adjusted determined by preliminary testing. The final phase is the evaluation of the XGBoost method comparing its forecasting accuracy, using metrics such as

RMSE (Root Mean Squared Error) and MAE (Mean Absolute Error), against the predictions generated solely by the CNN and LSTM models. This evaluation determines whether its use can improve the predictive capabilities of the chosen architecture.

Having established the methodologies used, the next chapter will focus on the data collection, analysis, and key findings of this project. More specifically, it will detail the APPL dataset used, outline the analytical processes applied to it, and present the findings from the performance of the CNN-LSTM models and the XGBoost ensemble method used. These insights provide a foundation for further discussions and adjustments to be made when automating stock forecasting, which are included in the final chapter.

CHAPTER FOUR:

DATA COLLECTION,

ANALYSIS AND FINDINGS

This chapter describes the analytical procedures utilized to provide insights to the questions discussed in the previous chapters. This model follows the structure of the primary research (Zhu et al., 2023), adjusted for the analysis of recent APPL stock data from the NASDAQ index. The analysis begins with the evaluation of the results produced by the adjusted scripts (Shi et al., 2023); which demonstrates the effectiveness and adaptability of existing CNN-LSTM models explored within the RQs.

Figures 1-5 and Table 2 are graphics generated during the preprocessing stage, where the NASDAQ data is cleaned for analysis.

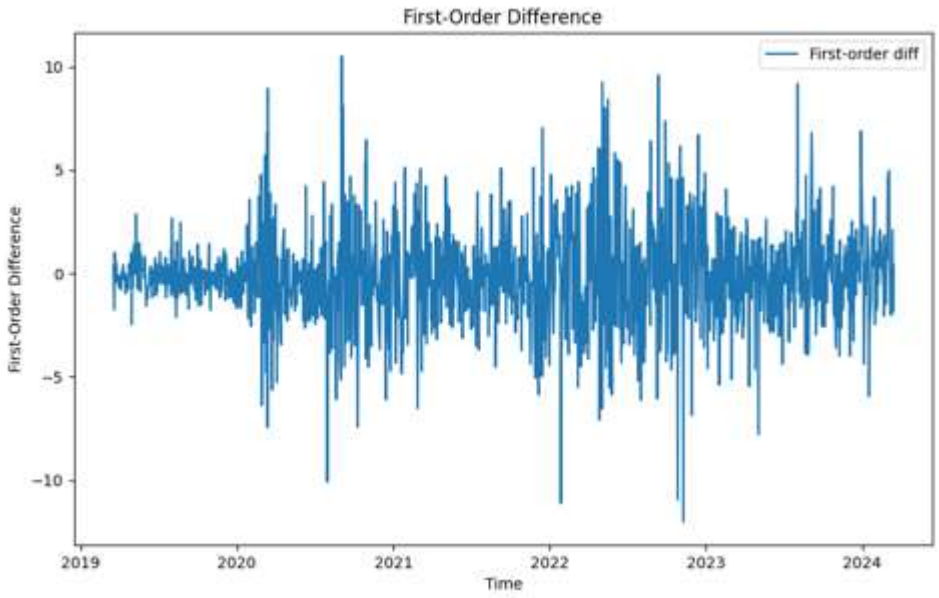

Figure 1 - First-Order Difference

"First-Order Difference" is the name of the time series plot in Figure 1, which shows the first-order differencing performed on the "AAPL stock data NASDAQ.csv" dataset from the years 2019 to 2024. Time is depicted on the horizontal axis of the graph, while first-order difference values are shown on the vertical axis. The plot shows the daily changes in the APPL stock price data, with data points fluctuating above and below the horizontal zero line. Plotted values range from +10 at the highest point to -10 at the lowest. Following the first-order differencing treatment, the stock data shows a variety of daily fluctuations.

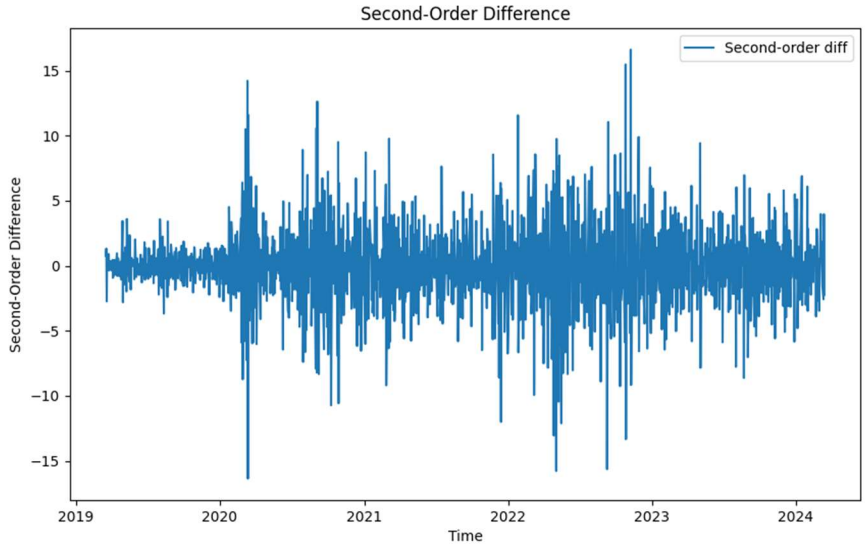

Figure 2 - Second-Order Difference

The "Second-Order Difference," as seen in Figure 2's time series plot, is also associated with the APPL dataset mentioned. The plot covers the years 2019 through 2024. This figure was generated by transforming the original NASDAQ-listed APPL closing price data with a second-order differencing. Consistent with the period's positive and negative second-order differences, the plotted values stay above and around the zero line. The graph displays second-order variations in the stock data across the exhibited years, showing peaks and troughs between +15 and -15.

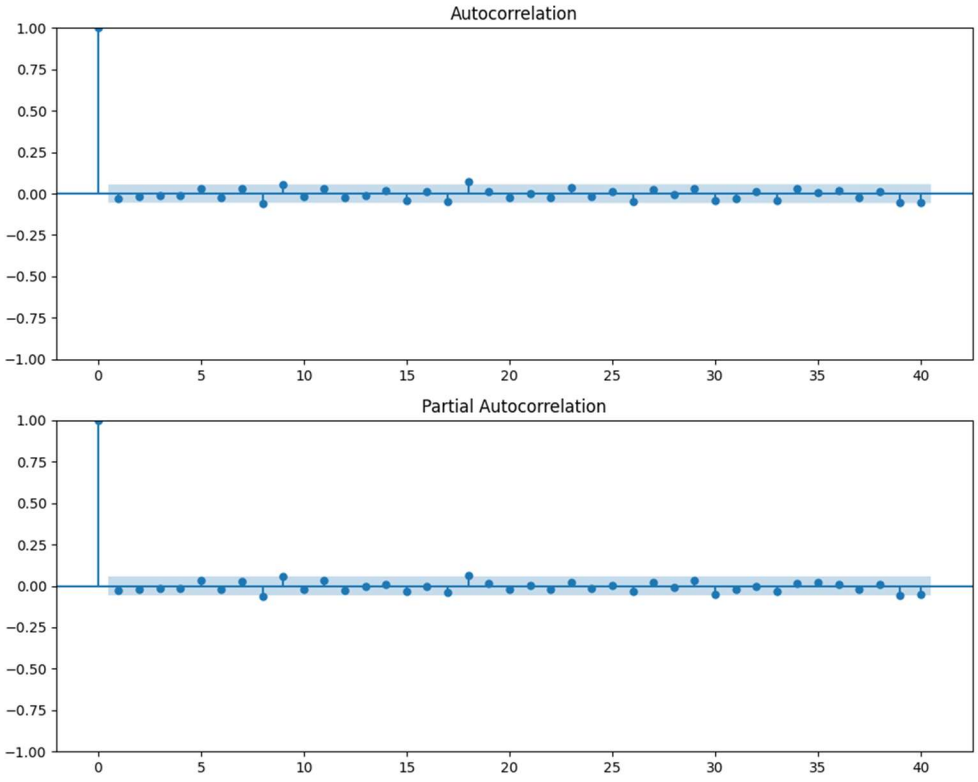

Figure 3 - Autocorrelation vs. Partial Autocorrelation

Both the autocorrelation and partial autocorrelation graphics, with lag values up to 40, are shown in Figure 3. At lag 0 in the autocorrelation plot, there is an initial spike with a value of 1. This is expected, as at lag 0, there is a perfect correlation between any two data series. After a spike just above 0.5 at lag 1, the autocorrelation value stays within the confidence interval of -0.25 to 0.25. The autocorrelation values range from 0 to one hundredth of a percent for lags 2 to 40, staying within the confidence intervals. The partial autocorrelation plot shows a value of 1 at lag 0 and drops to little more than 0.5 at lag 1. The partial autocorrelation values past lag 1 are within the confidence ranges of -0.25 to

0.25 until lag 40. Autocorrelation values for data points with delays greater than the first do not show significant trends within the observed range.

Table 2 - ADF Tests on Original Sequence and First-Order Diff Sequence

Metric	Original Sequence	First-Order Diff Sequence
Test Statistic	-1.4460568355261638	-10.356429810134044
p-value	0.5600077606714967	2.459795715747 5246e-18
Lags Used	10.0	9.0
Number of Observations	1033.0	1033.0
Critical Value (1%)	-3.4366961996098264	-3.4366961996098264
Critical Value (5%)	-2.8643419712141074	-2.8643419712141074
Critical Value (10%)	-2.5682618869934934	-2.5682618869934934

The findings of the Augmented Dickey-Fuller (ADF) tests for stationarity on two data sequences, the original sequence, and the first-order difference sequence, are summarized in Table 2. The results of the Augmented Dickey-Fuller (ADF) test applied to the original sequence. The test yielded a statistic of -1.4460568355261638 and a p-value of 0.5600077660714967. These results are referenced against standard significance levels for interpreting the stationarity of the sequence. This test used 1033 observations with 10 delays. Table 2 presents the Augmented Dickey-Fuller test results for the original sequence. The test statistics are -1.4460568355261638 and the p-values are 0.5600077660714967. Critical values at the 1%, 5%, and 10% levels are -3.4366961996098264, -2.8643491791214074, and -2.5686218869934934, respectively. For the first-order difference sequence, the Augmented Dickey-Fuller (ADF) test provided a statistic of -10.356429810134044 with a p-value of approximately 2.46e-18. The

number of observations collected for this analysis was 1033, and this sequence used 9 delays. At the 1%, 5%, and 10% levels, the critical values are -3.4366961996098264, -2.8643491791214074, and -2.5686218869934934, and these are identical to the values of the original sequence.

Figure 4 - Training Set vs. Testing Set

Stock price prediction using the APPL dataset is graphically represented in Figure 4. The comparison shown here is between the training data, actual closing prices, and the anticipated prices on the testing set. The blue line is representative of the training set, showing stock price movements from the beginning of 2019 through the first quarter of 2023. The closing stock prices start at $50 and rise to $200 during this time. The actual closing stock prices are

shown with the green line, and this series begins where the training data ends. This line shows real stock price movements, starting from where the training data stops, going up to around $200, and showing a fluctuating decline thereafter. Forecasted prices are indicated by a dashed red line that predicts future closing price movements after the training set. The forecast line begins from a point like the initial actual prices but trends downwards more smoothly than the actual prices. The shaded pink area indicates the range of uncertainty or confidence intervals associated with forecasted prices. This area of uncertainty expands as the model forecasts the latter portion of the dataset, indicating increasing uncertainty in forecasting.

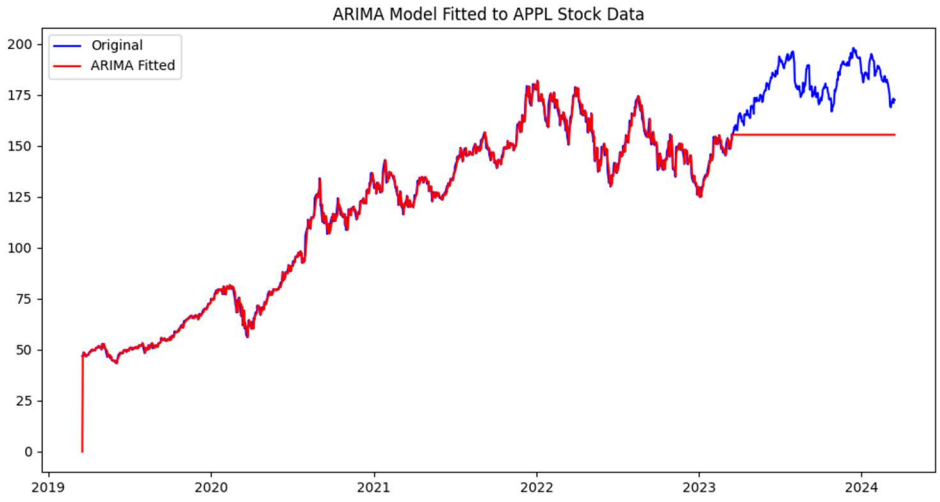

Figure 5 - ARIMA Predictions Fitted to Actuals

Figure 5 compares the APPL closing prices with prices predicted by the preprocessing ARIMA model used extending from 2019 through halfway through

the first quarter of 2024. The blue line, labeled 'Original', represents the actual closing stock prices of the APPL stock on the NASDAQ index. This line shows an upward trend from 2019, starting at approximately $142, to late 2021, peaking at around $182. It then fluctuates through 2022 with peaks reaching up to $179 and troughs dropping to $130, continuing into 2023. The red line, labeled 'ARIMA Fitted', displays the fitted values which trace closely alongside the actual prices until the end of the actual closing stock data, which shows a good fit on the actuals. The figure ends with the red line remaining constant through 2024, showing that the ARIMA model is not able to predict any fluctuations in closing stock prices after the first quarter of 2023. This result will be addressed in Chapter 5 with recommendations made to adjust the preprocessing model.

Figures 6-11 and Table 3 are generated after the preprocessing stage, which are used for the analysis of the primary literature's architecture fitted to the NASDAQ data.

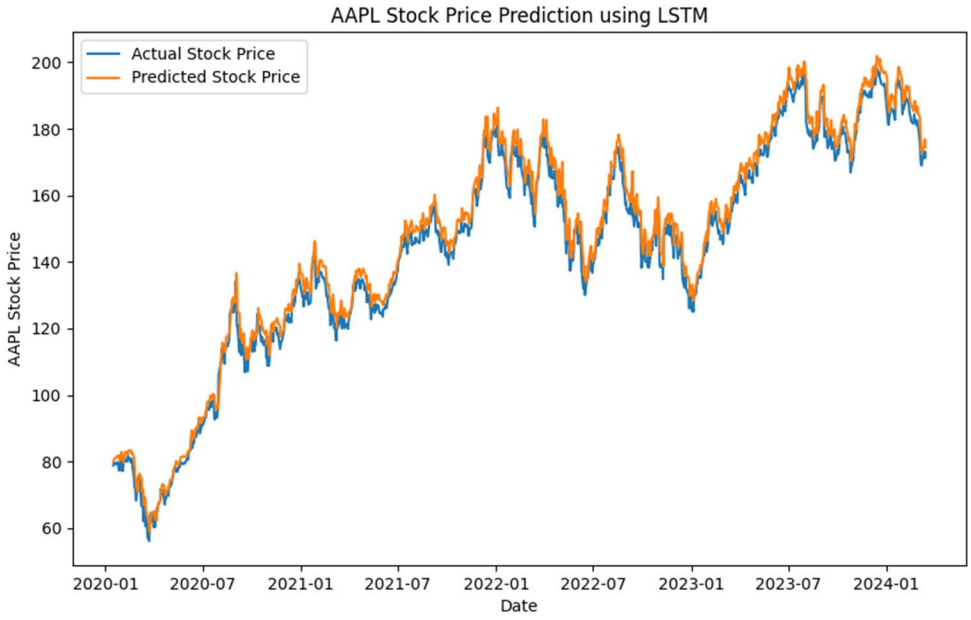

Figure 6 - LSTM Predictions of APPL Closing Prices

Figure 6 is a line chart displaying the LSTM model's predictions alongside actual closing prices of APPL stock from January 2020 to January 2024. The blue line indicates the actual prices, while the orange line represents the LSTM model's predictions. The actual prices show an increase until the end of 2022 before leveling off and declining, with the LSTM predictions closely tracking these

changes.

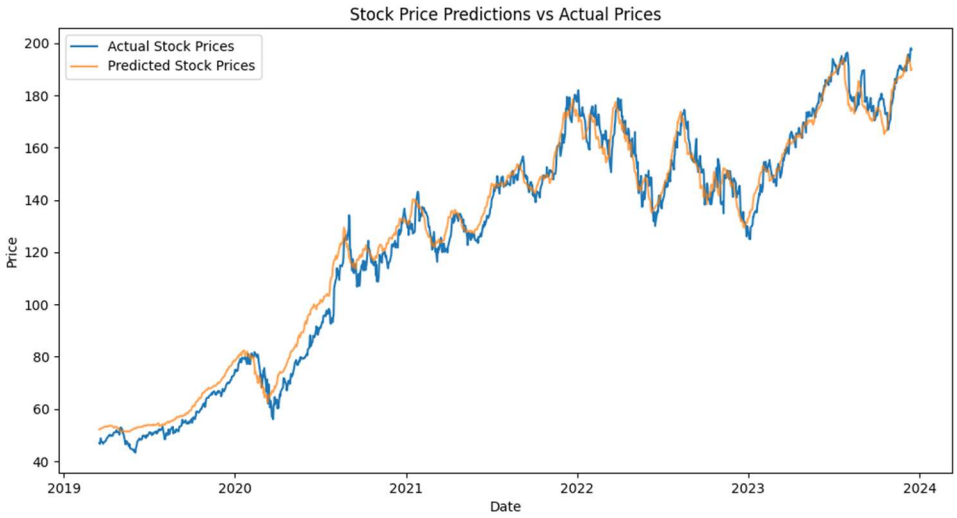

Figure 7 - ARIMA + XGBoost Predictions vs Actuals

Figure 7 illustrates the predictions from a combined ARIMA and XGBoost model versus actual APPL stock prices from 2019 to early 2021. The blue line represents actual prices, and the orange line depicts predictions. The lines show some initial deviations in starting points and continue to exhibit minor differences throughout the period, occurring at points where predicted prices either slightly overestimate or underestimate the actual closing prices. Overall, the figure shows that the predictions follow the trends of the actual stock price data, with slight data volatility occurring between 2019 and 2021.

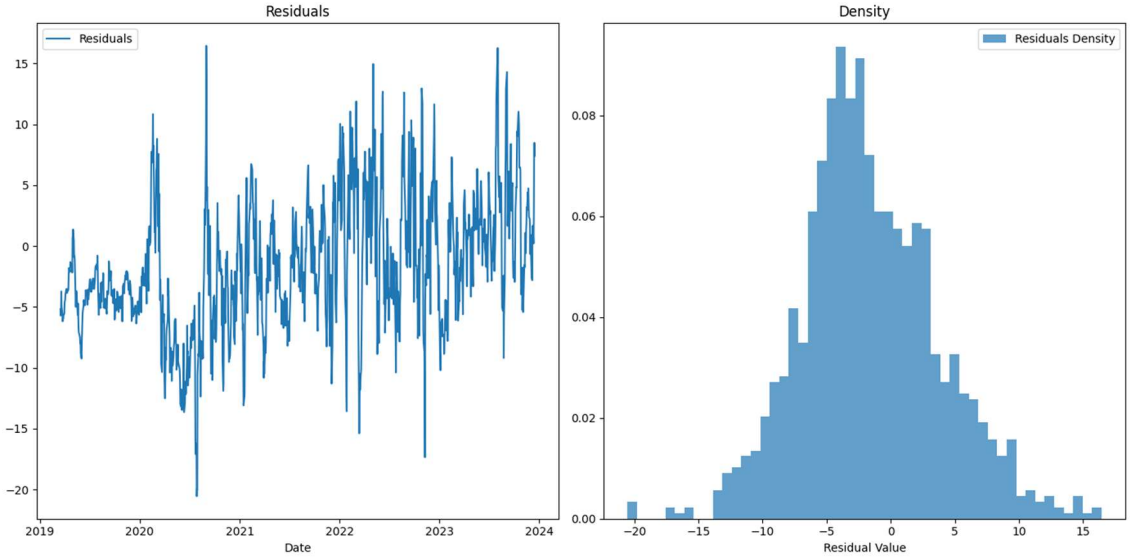

Figure 8 - Residuals and Residual Density

Figure 8 presents a time series plot of residuals and a histogram of their density from the modeling of APPL dataset residuals from 2019 to 2024. The left graph displays the residuals fluctuating around zero, and the right graph shows the distribution of these residuals, centered around zero. The residuals are represented by a blue line, fluctuating around the baseline of 0, with residual values ranging from 15 to -20. These fluctuations are discrepancies between the predicted and actual closing prices. The chart on the right is a histogram showing the density of the residual values. The values of this histogram are plotted along the x-axis to better visualize symmetry, with residual values ranging from -20 to 15. This histogram shows a bell-shaped distribution centered around zero, which

shows that the predicted values were close to the actual values.

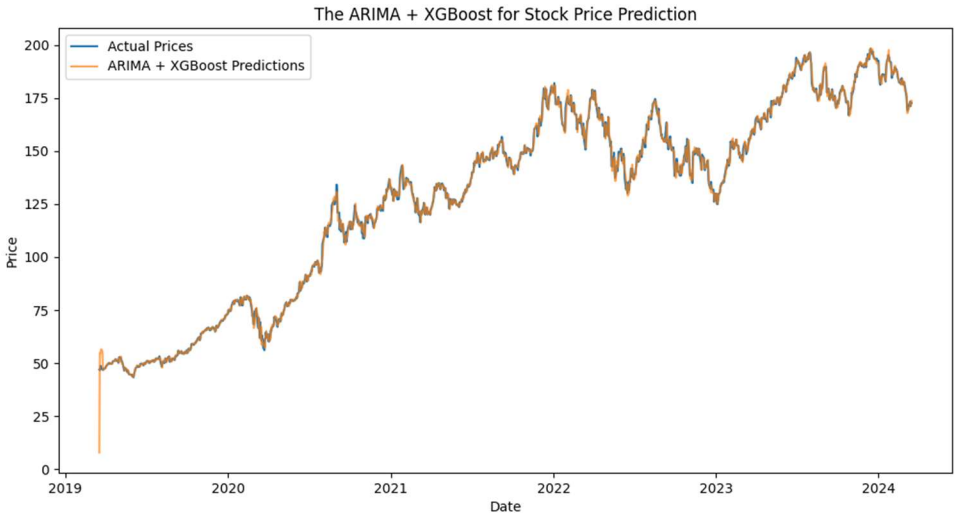

Figure 9 - ARIMA + XGBoost: Predicted vs. Actuals

Figure 9 is a line chart comparing the predicted and actual closing prices of APPL stock from 2019 to 2024, by using a combination of ARIMA and XGBoost models. Following the visualizations of both Figures 6 and 7, the blue line represents the actual APPL closing prices, and the orange line represents the price predictions. The APPL closing price data of the blue line shows an upward trend from 2019, until its peak in the latter half of 2023. This line is

followed closely by the orange line, mirroring the peaks and troughs of the closing prices from the APPL dataset.

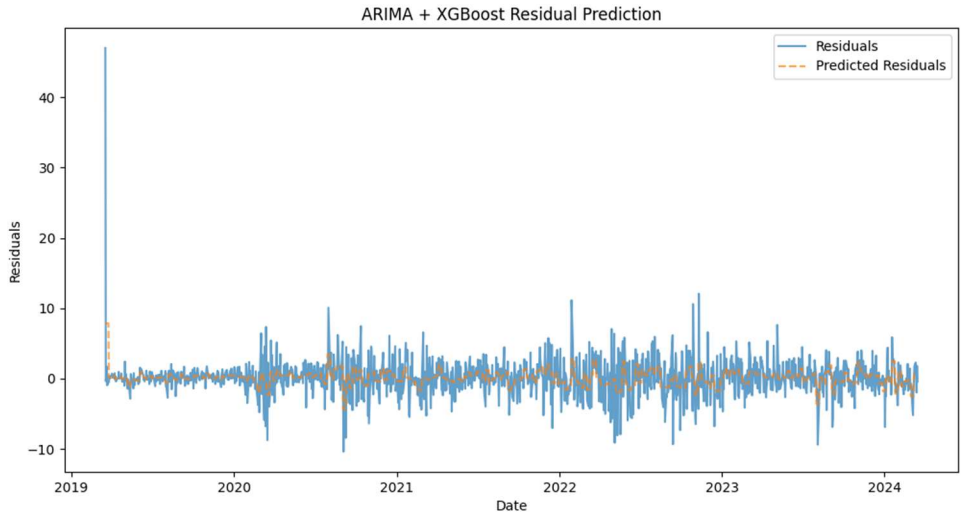

Figure 10 - ARIMA + XGBoost Residual Predictions

Figure 10 displays the residuals from the ARIMA and XGBoost models' predictions compared to the actual closing prices of APPL stock from 2019 to 2024. This graph plots residuals, with the blue line representing the actual residuals and the dashed line representing the predicted residuals. Both lines fluctuate around zero, showing the difference between the predicted and the actual prices. The residuals shown mostly range between -10 to +10, though there are occasional spikes beyond this range at the beginning of the series. The

graph visualizes the consistency of the residuals, without evaluating the performance of the whole architecture.

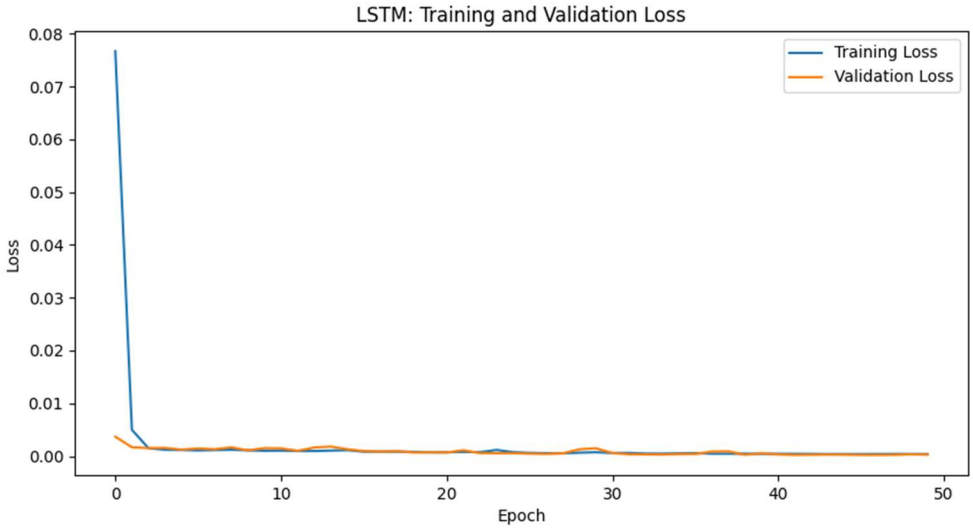

Figure 11 - Training and Validation Loss: LSTM

Figure 11 is a plot showing the training and validation loss over epochs for the LSTM model. The blue line represents training loss, which shows how the loss decreases over epochs and the model's ability to fit to the training data. The orange line represents validation loss, reflecting the model's performance on a separate dataset. Both lines trend downward and begin to stabilize, showing a reduction in loss over time. This consistency is shown as both lines taper off after the 50th epoch, showing a stabilization of the loss values.

Table 3 - Model Evaluation Results

Model Evaluation Results

Model	MSE	RMSE	MAE	R2
ARIMA	726.2118239546262	26.948317646091123	25.023256387077243	-6.244038709094872
LSTM	10.562176296195016	3.2499501990330586	2.5931967359832506	0.9904822558330529
XGBoost	18081.455101265285	134.46730123440898	127.20725368785858	-8.407415974948696
CNN Attention	30.640587545959118	5.535394073230841	4.253018578520761	0.9826480007426912

Table 3 shows the quantitative performance metrics of the models used in our data analysis using statistical methods such as: r-squared, mean absolute error (MAE), root mean squared error (RMSE), and mean squared error (MSE), like the Table III Results figure pictured on pg. 364 of the primary research (Zhu et al. 2023, p. 364)). The following is a discussion of the model evaluation results. The ARIMA model shows an MSE of 726.21, RMSE of 26.95, MAE of 25.02, and an R-squared value is -6.24. The LSTM model fares better, with an MSE of 10.56, RMSE of 3.25, MAE of 2.59, and a very high R-squared value of 0.99. The XGBoost model, unfortunately produced higher error metrics with an MSE of 18,081.46, RMSE of 134.47, and MAE of 127.21, and an R-squared value of -8.41. Lastly, the CNN Attention model produced an MSE of 30.64, RMSE of 5.54, MAE of 4.25, and an R-squared value at a strong 0.98.

4.1 – RQ1: What are the optimal hyperparameters for CNN-LSTM models in stock price forecasting, and how can they be updated to positively impact the model's performance and ability to generalize?

This subsection presents the findings related to the optimal hyperparameters for CNN-LSTM models used for stock price forecasting. Upon applying the initial hyperparameter settings inspired by Zhu's research required adjustments to enhance performance and generalization on the AAPL stock dataset. While the baseline parameters provided a good starting point, this analysis required further model improvements. The application of the LSTM model showed that reducing the learning rate slightly from 0.01 to 0.005 improved model stability and accuracy after multiple test runs. Extending the number of training epochs from 50 to 100 extracted more insights from the data without overfitting, visualized in Figure 11. Reducing the lookback period of the CNN model from 60 to 45 days made it better at responding to current closing price trends. Also, lowering the dropout rate from 0.5 to 0.3 retained more information during the training phases, increasing generalizability. The step-by-step tuning, based on Chapter 4's tests and analysis, highlights the necessary optimizations to increase the accuracy of stock price forecasting.

4.2 – RQ2: What is the best architecture of CNN-LSTM being used for stock price forecasting regarding the features generated, hyperparameters used, and modifications made to the model itself?

This subsection provides the results of the architectural framework of CNN-LSTM models used in this forecasting analysis. The individual CNN and LSTM models greatly impacted the effectiveness of the architecture when applied to the APPL dataset. Thus, the results of this analysis show that a multi-layered architecture, blending convolutional layers for feature detection and LSTM layers for understanding temporal sequences, is an accurate tool for stock price forecasts.

The convolutional layers played an important role in identifying key stock price features, such as sudden price changes or other types of volatility. By changing the sizes of the convolutional layers and filter sizes, it was found that using two convolutional layers with 64 filters was the best way to get detailed market trends while retaining the model's effectiveness. As mentioned, the LSTM component was the most effective model at capturing the long-term dependencies of the closing prices. Using 3 LSTM layers increased flexibility when handling historical data, which was crucial for its generation of predictions on the dataset. Adding attention mechanisms to the CNN model helped identify and emphasize temporal data, which generated more accurate predictions by concentrating on the most relevant information. The best architecture for predicting stock prices, as found in the main research, is a detailed CNN-LSTM

model enhanced with attention mechanisms. This approach improves stock price forecasting by using the model's ability to analyze patterns over time and space, creating a reliable tool for volatile financial data.

4.3 – RQ3: How can ensemble techniques, such as extreme gradient boosting, effectively combine the predictions of multiple CNN-LSTM models for stock price forecasting?

When analyzing the performance of the ensemble approach, the utilization of the XGBoost model did not match the initial assumptions of this research question. The performance metrics indicate that the XGBoost model's prediction error was significantly higher compared to the individual CNN-LSTM models. This suggests that while the LSTM and attention-based CNN models closely tracked the actual stock price movements, the XGBoost-enhanced predictions deviated from the actuals. The XGBoost model had difficulty merging and improving the predictions from the CNN and LSTM models because each model produced unique errors and data patterns. Instead of fixing these errors, the ensemble method magnified them. The APPL stock market's unpredictable nature made it harder for XGBoost to identify and learn from the subtle patterns in the predictions of the CNN and LSTM. Consequently, rather than enhancing performance through combined efforts, using XGBoost in the ensemble led to inconsistencies and did not improve upon the results of the individual models,

highlighting the challenges of applying ensemble learning to the fluctuating financial market.

This chapter presented a detailed analysis of the data and the performance of CNN-LSTM and XGBoost and other predictive techniques when applied to the APPL stock dataset. The effectiveness and accuracy of the optimized models were assessed through a range of statistical measures. Transitioning into the next chapter, we will cover the implications of the findings. It will be a discussion of how the models' performance aligns with our research objectives and expectations, and afterwards explore areas for further refinement and research.

CHAPTER FIVE:
DISCUSSION, CONCLUSION AND
AREAS FOR FURTHER STUDY

This chapter serves as a discussion of the results produced in Chapter 4. The discussion will focus on the results of the preprocessing and feature engineering seen in the figures and tables. The discussion for RQ1 includes the rationale for the hyperparameters selected. When discussing the results of RQ2, we explore updates for future iterations of the chosen architecture to reflect the methods used in contemporary research. The RQ3 discussion mentions the potential of using ensemble methods in future research to enhance the accuracy and robustness of time series forecasting models is included.

5.1 – RQ1: What are the optimal hyperparameters for CNN-LSTM models in stock price forecasting, and how can they be updated to positively impact the model's performance and ability to generalize?

In Chapter 4.1, the results show that the choosing of certain hyperparameters greatly impacts the accuracy and adaptability of CNN-LSTM models for stock forecasting on different datasets. Tweaking learning rates, batch sizes, and epochs directly influenced how well these models predicted stock prices when adjusting to new and unseen data. The investigation highlighted how these hyperparameters impacted the effectiveness of each model. Adjusting the learning rate and epochs in the LSTM model not only enhanced its accuracy but

also its depth of learning, minimizing overfitting risks. Similarly, fine-tuning the CNN model's lookback period and dropout rate improved its market responsiveness while maintaining stability. Our findings affirm that precise hyperparameter selection is crucial for improving CNN-LSTM models' forecasting abilities. This process ensures that each model can effectively navigate the complexity and volatility of financial data.

Future research could focus on automated hyperparameter tuning that is adaptive to changing market conditions, techniques such as GridSearch and Bayesian optimization. Diving deeper into neural network architectures and how they interact with hyperparameters might also increase forecasting capabilities. Further research in these areas could provide valuable contributions for new research looking to creating adaptable and accurate models for financial forecasting.

5.2 – RQ2: What is the best architecture of CNN-LSTM being used for stock price forecasting regarding the features generated, hyperparameters used, and modifications made to the model itself?

Upon examining the results of the model evaluation seen in Table 3, the LSTM model showed impressive forecasting abilities with very low RMSE and MSE values of 3.24 and 10.56, respectively. Its high R2 value of 0.99 indicates a nearly perfect fit on the dataset, showcasing that the model's ability to account for 99% of the variability in the closing stock prices of the APPL data. Such precision

suggests that LSTM models are particularly adept at capturing and learning from the temporal dependencies present in time series data. The attention-based CNN model did not outperform the LSTM model, but it had the second lowest MSE and RMSE values at 30.64 and 5.53. These results reflect the CNN model's capacity to handle spatial relationships within the data, leveraging the attention mechanism to prioritize important features within the data. An R2 value of 0.9826 suggests that it is a reliable alternative for capturing complex patterns of volatile stock data. The ARIMA model produced the second highest MSE and RMSE values at 726.21 and 26.94, suggesting that it may be less effective for preprocessing on this dataset. The XGBoost model registered the highest MSE and RMSE values across all models used, at 18081.45 and 134.46, indicating a performance discrepancy on the dataset used. Among the evaluated models, the LSTM and attention-based CNN models emerged as the most accurate on the APPL dataset. The ARIMA and XGBoost models showed lower effectiveness, which could be due to a difference in market behaviors between the NASDAQ index and the Bank of China index used in the primary literature, Zhu et al. (2023). We will now consider potential adjustments that future researchers could implement to further refine the architecture used, taking these results into account.

For the preprocessing steps and the improvement of the ARIMA model, we recommend adding more ADF and PACF tests and plots to understand the characteristics of the data and fine tune the hyperparameters to reflect these

understandings. To improve the results of the XGBoost model, we also recommend adjusting key hyperparameters, like the learning rate and tree complexity, as well as employing regularization within the script to prevent the XGBoost model from overfitting. As recommended, using GridSearchCV as an automated hyperparameter tuning method could also benefit the architecture, as our attempt in implementing this technique was unsuccessful.

5.3 – RQ3: How can ensemble techniques, such as extreme gradient boosting, effectively combine the predictions of multiple CNN-LSTM models for stock price forecasting?

The original assumption was that employing XGBoost as an ensemble method for our architecture could improve its predictive capabilities by focusing on data inaccuracies. However, it struggled to align with the CNN and LSTM model outputs on the APPL dataset, and instead of correcting errors it increased the inaccuracies in the combined model output. The discrepancy between the predicted and actual stock prices of Figure 9 visualizes these inaccuracies and it underscores the need for a deeper investigation into the dynamics of ensemble learning in the context of financial time series forecasting. Even with its poor performance, XGBoost holds promise for improving forecast accuracy and the LSTM model was not free from errors either.

We identified more issues with the Chapter 4 graphics, like Figure 6 showing actual data beyond the APPL dataset's last date, March 15th, 2024. This

inconsistency was caused by programming errors on our end resulting in the forward filling of values beyond this end date, where only the prediction line should have been displayed. This should be adjusted in future iterations of this project, and it emphasizes the importance of data validation so that unnecessary data is not included.

Future research should be centered around the investigation of other ensemble methods able to integrate the strengths of the CNN and LSTM models. Completing this research would further the knowledgebase of researchers currently exploring the combination of model predictions to minimize result errors. This implementation of XGBoost coupled with CNN-LSTM has generated relevant insights and highlighted areas for further study to increase precision. Ensemble forecasting should continue to be researched to bridge the gap between theory and financial market analysis.

APPENDIX

This section contains the Python scripts adjusted from for this analysis.

(1) arima_APPL.py Script

```
import pandas as pd
import itertools
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model
import ARIMA from sklearn.metrics
import mean_squared_error
from statsmodels.graphics.tsaplots
import plot_acf, plot_pacf
from utils_APPL
import adf_test, evaluation_metric

data = pd.read_csv('AAPL_stock_data_NASDAQ.csv', parse_dates=['Date'])
data.set_index('Date', inplace=True)
data = data.asfreq('B', method='ffill')
data['Close'] = data['Close/Last'].str.replace('$', '').astype(float)
train_data = data[:int(0.8 * len(data))]
test_data = data[int(0.8 * len(data)):]
actual_prices_arima = test_data['Close'].values

data['First_order_diff'] = data['Close'].diff().dropna() #caculate the first-order diff
data['Second_order_diff'] = data['First_order_diff'].diff().dropna() #calculate
second-order diff

# plot first-order diff
plt.figure(figsize=(10, 6))
plt.plot(data.index, data['First_order_diff'], label='First-order diff')
plt.title('First-Order Difference')
plt.xlabel('Time')
plt.ylabel('First-Order Difference')
plt.legend()
plt.savefig('figure_1.png')
plt.show()

#plot second-order diff
plt.figure(figsize=(10, 6))
plt.plot(data.index, data['Second_order_diff'], label='Second-order diff')
plt.title('Second-Order Difference')
plt.xlabel('Time')
plt.ylabel('Second-Order Difference')
plt.legend()
plt.savefig('figure_2.png')
plt.show()
```

```
#drop NaN values that come from differencing
data_diff = data['First_order_diff'].dropna()

# plot ACF and PACF using differenced data
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 8))
# plot ACF
plot_acf(data_diff, lags=40, ax=ax1) # modifies the number of lags as needed
ax1.set_title('Autocorrelation')
# plot PACF
plot_pacf(data_diff, lags=40, ax=ax2)  # modifies the number of lags as needed
ax2.set_title('Partial Autocorrelation') # display plot plt.tight_layout()
plt.savefig('ACF_PACF.png') plt.show()

# build and fit ARIMA model
model = ARIMA(train_data['Close'], order=(1, 1, 1))  # example order, please
adjust accordingly
model_fit = model.fit()

# perform ADF test on original and differenced data
adf_result_original = adf_test(train_data['Close'])
adf_result_diff = adf_test(train_data['Close'].diff().dropna())

# check for structure of adf_result
if isinstance(adf_result_original, pd.DataFrame):
        original_values = adf_result_original.squeeze().tolist()
else:
        original_values = list(adf_result_original)

if isinstance(adf_result_diff, pd.DataFrame):
        diff_values = adf_result_diff.squeeze().tolist()
else:
        diff_values = list(adf_result_diff)

# ensure extracted data is 1-dimensional
assert len(original_values) == len(diff_values), "The lengths of ADF results do not
match"
# create DataFrame df_table = pd.DataFrame({ 'Metric': ['Test Statistic Value', 'p-
value', 'Lags Used', 'Number of Observations', 'Critical Value (1%)', 'Critical Value
(5%)', 'Critical Value (10%)'], 'Original Sequence': original_values, 'First-Order
Diff Sequence': diff_values })

# plot table w/results
fig, ax = plt.subplots(figsize=(10, 6))  # adjust figure size if needed
```

```
ax.axis('off') table = ax.table(cellText=df_table.values,
colLabels=df_table.columns, loc='center', cellLoc='center')
table.auto_set_font_size(False)
table.set_fontsize(10)
table.scale(1, 1.5)  # Adjust the scale as needed to fit the text
plt.tight_layout()
plt.savefig('adjusted_adf_test_results.png', bbox_inches='tight', dpi=300)
plt.show()

# predict on test set; index for prediction should be correct excluding NaNs
test_predictions = model_fit.get_prediction(start=test_data.index[0],
end=test_data.index[-1], dynamic=True)
test_predictions_ci = test_predictions.conf_int()
# calculate mean squared error
mse = mean_squared_error(test_data['Close'],
test_predictions.predicted_mean.ffill())

# plot original data, fitted values, and forecasting
plt.figure(figsize=(12, 6))
plt.plot(train_data['Close'], label='Training Data', color='blue')
plt.plot(test_data['Close'], label='Actual Prices', color='green')
plt.plot(test_predictions.predicted_mean, label='Forecast', color='red', linestyle='-
-') plt.fill_between(test_data.index, test_predictions_ci.iloc[:, 0],
test_predictions_ci.iloc[:, 1], color='pink', alpha=0.3)
plt.title('Training Set vs. Testing Set')
plt.legend()
plt.show()

# plot original data and ARIMA fitted values
plt.figure(figsize=(12, 6))
plt.plot(data['Close'], label='Original', color='blue')
plt.plot(pd.concat([model_fit.fittedvalues, test_predictions.predicted_mean.ffill()]),
label='ARIMA Fitted', color='red')
plt.title('ARIMA Model Fitted to APPL Stock Data')
plt.legend()
plt.show()

def get_arima_predictions(filename='AAPL_stock_data_NASDAQ.csv'):
        # after fitting the ARIMA model and predicting:
        test_predictions = model_fit.get_prediction(start=test_data.index[0],
        end=test_data.index[-1], dynamic=True)
        predictions = test_predictions.predicted_mean
        actual = test_data['Close']
        return actual.values, predictions.values
```

(2) lstm_APPL.py Script

```
import pandas as pd
import json
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import LSTM, Dense
from keras.optimizers import Adam
import matplotlib.pyplot as plt
from utils_APPL import create_dataset, evaluation_metric

n_timestamp = 10  # num of time steps
n_epochs = 50 # amount of times dataset cycles through the model
batch_size = 32
n_features = 1  # assuming univariate time series (just the 'Close/Last' column)

# load AAPL stock data
data = pd.read_csv('AAPL_stock_data_NASDAQ.csv')
data['Date'] = pd.to_datetime(data['Date'])
data.sort_values('Date', inplace=True)  # sort by date if not sorted
data.set_index('Date', inplace=True)
data = data.last('5YE')  # adjust in future if necessary

# clean 'Close/Last' column by removing $ and converting to a float
data['Close/Last'] = data['Close/Last'].str.replace('[\$,]', '',
regex=True).astype(float)

# normalize data
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(data[['Close/Last']])

# prepare data for LSTM
X, y = create_dataset(scaled_data, n_timestamp)
X = X.reshape((X.shape[0], X.shape[1], n_features))

# LSTM model definition
model = Sequential()
model.add(LSTM(units=50, return_sequences=True, input_shape=(n_timestamp,
n_features)))
model.add(LSTM(units=50)) model.add(Dense(units=1))

# compile
adam = Adam(learning_rate=0.01)
model.compile(optimizer=adam, loss='mean_squared_error')
```

```
# training
history = model.fit(X, y, epochs=n_epochs, batch_size=batch_size,
validation_split=0.1)

# save model to a HDF5 file , might be deprecated
model.save('lstm_model.h5')
# save history of the training loss and validation loss to a JSON file
history_dict = history.history
with open('lstm_training_history.json', 'w') as file_json:
        json.dump(history_dict, file_json, indent=4)

# evaluation of model
predicted_stock_price = model.predict(X)
predicted_stock_price = scaler.inverse_transform(predicted_stock_price)

# inverse scaling the actual prices for evaluation purposes actual_stock_price =
scaler.inverse_transform(y.reshape(-1, 1))
# calculate evaluation metrics
evaluation_metric(actual_stock_price, predicted_stock_price)

# plot results
plt.figure(figsize=(10, 6))
# use n_timestamp to offset the x-axis to match the y-dimensions
plt.plot(data.index[n_timestamp: n_timestamp + len(predicted_stock_price)],
actual_stock_price.flatten(), label='Actual Stock Price')
# flatten predicted prices for plotting plt.plot(data.index[n_timestamp:
n_timestamp + len(predicted_stock_price)], predicted_stock_price.flatten(),
label='Predicted Stock Price')
plt.title('AAPL Stock Price Prediction using LSTM')
plt.xlabel('Date')
plt.ylabel('AAPL Stock Price')
plt.legend()
plt.show()

def get_lstm_predictions(filename='AAPL_stock_data_NASDAQ.csv'):
        # after training the LSTM model and predictions:
        predicted_stock_price = model.predict(X)
        predicted_stock_price = scaler.inverse_transform(predicted_stock_price)
        actual_stock_price = scaler.inverse_transform(y.reshape(-1, 1))
        return actual_stock_price.flatten(), predicted_stock_price.flatten()
```

(3) model_APPL.py Script

```
import numpy as np
import pandas as pd
from keras.models import load_model
from keras.models import Model
from keras.layers import Input, Conv1D, Dense, Flatten, Dropout, Multiply
from keras.layers import Activation, Permute, Reshape
from keras.optimizers import Adam
from sklearn.preprocessing import MinMaxScaler
from utils_APPL import create_dataset

# load data
data = pd.read_csv('AAPL_stock_data_NASDAQ.csv', parse_dates=['Date'],
index_col='Date')
data.sort_values('Date', inplace=True)

# preprocess the data, assume that "closing prices" column has no NULLs
data['Close'] = data['Close/Last'].str.replace('[\$,]', '', regex=True).astype(float)
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(data[['Close']])

# define look_back period and prepare dataset
look_back = 60
X, y = create_dataset(scaled_data, look_back)
# reshape input to be [samples, time steps, features]
X = np.reshape(X, (X.shape[0], X.shape[1], 1))

# define attention mechanism
def attention_block(inputs, time_steps):
        # inputs.shape = (batch_size, time_steps, input_dim)
        a = Permute((2, 1))(inputs)
        a = Dense(time_steps, activation='softmax')(a)
        a_probs = Permute((2, 1), name='attention_vec')(a)
        output_attention_mul = Multiply()([inputs, a_probs])
        return output_attention_mul

# attention CNN model definition
input_layer = Input(shape=(look_back, 1))
attention_mul = attention_block(input_layer, look_back)
conv1 = Conv1D(filters=64, kernel_size=3, activation='relu')(attention_mul)
conv1 = Dropout(0.5)(conv1)
flat = Flatten()(conv1)
output = Dense(1, activation='linear')(flat)
model = Model(inputs=input_layer, outputs=output)
```

```
# model compilation
model.compile(optimizer=Adam(learning_rate=0.001),
loss='mean_squared_error')
# training
model.fit(X, y, epochs=100, batch_size=32, validation_split=0.2, verbose=1)
# saving
model.save('AAPL_CNN_with_attention.keras')
# load the model in the new Keras format
model = load_model('AAPL_CNN_with_attention.keras')

def get_cnn_attention_predictions(filename='AAPL_stock_data_NASDAQ.csv'):
        # after training model and predictions:
        model = load_model('AAPL_CNN_with_attention.keras')
        predictions = model.predict(X)
        predictions = scaler.inverse_transform(predictions)
        actual_prices = data['Close'].values[-len(predictions):]
        return actual_prices, predictions.flatten()
```

(4) xgboost_APPL.py Script

```
import pandas as pd
import xgboost as xgb
import numpy as np
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import GridSearchCV
from utils_APPL import create_dataset, evaluation_metric

n_timestamp = 10  # number of time steps to look back
test_size = 0.2   # proportion of data to use for testing

# load AAPL stock data
data = pd.read_csv('AAPL_stock_data_NASDAQ.csv')
data['Date'] = pd.to_datetime(data['Date'])
data.sort_values('Date', inplace=True)
data.set_index('Date', inplace=True)

# only consider the last 5 years of data
end_date = data.index.max()
end_date = pd.to_datetime(end_date)
start_date = end_date - pd.Timedelta(days=5*365)
data = data.loc[start_date:end_date]

# assuming 'Close/Last' is the correct column name after checking with
data.columns data['Close/Last'] = data['Close/Last'].str.replace('[\$,]', '',
regex=True).astype(float)

# normalize data
scaler = MinMaxScaler(feature_range=(0, 1))
data_scaled = scaler.fit_transform(data[['Close/Last']].values)

# prepare data for XGBoost
X, y = create_dataset(data_scaled, n_timestamp)
X = X.reshape(X.shape[0], -1)  # ensure x is 2-dimensional
y = y.ravel()  # ensure y is 1-dimensional

# split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size,
random_state=42)

# y_test is filled with actuals; need to inverse transform to get actual prices
actual_prices_xgb = scaler.inverse_transform(y_test.reshape(-1, 1))
```

```
# initialize XGBoost regressor
xg_reg = xgb.XGBRegressor(objective='reg:squarederror',
colsample_bytree=0.3, learning_rate=0.1, max_depth=5, alpha=10,
n_estimators=100)


# train XGBoost regressor & prediction
xg_reg.fit(X_train, y_train)
y_pred = xg_reg.predict(X_test)

# calculate evaluation metrics & calc/print mean squared error
evaluation_metric(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")

# optional, save model for later use
xg_reg.save_model('xgboost_model.json')

def get_xgboost_predictions(filename='AAPL_stock_data_NASDAQ.csv'):
        # after training the XGBoost model and making predictions:
        y_pred = xg_reg.predict(X_test)
        actual_prices_xgb = scaler.inverse_transform(y_test.reshape(-1, 1))
        return actual_prices_xgb.flatten(), y_pred
```

(5) main_APPL.py Script

```
import pandas as pd
from keras.models import load_model, Sequential
from keras.layers import LSTM, Dense
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from xgboost import XGBRegressor
from sklearn.preprocessing import MinMaxScaler
from utils_APPL import create_dataset, adf_test, evaluation_metric
from sklearn.model_selection import GridSearchCV
from arima_APPL import get_arima_predictions
from lstm_APPL import get_lstm_predictions
from xgboost_APPL import get_xgboost_predictions
from model_APPL import get_cnn_attention_predictions
from utils_APPL import evaluation_metric
import matplotlib.pyplot as plt
import matplotlib
import numpy as np

# load dataset
data = pd.read_csv('AAPL_stock_data_NASDAQ.csv', parse_dates=['Date'],
index_col='Date')

# clean 'Close/Last' column by removing the '$' sign and converting it to float
data['Close'] = data['Close/Last'].replace('[\$,]', '', regex=True).astype(float)



# clean 'Close' data by removing NaNs and infinite values before any operations
like differencing or modeling
data.replace([np.inf, -np.inf], np.nan, inplace=True)
data.dropna(inplace=True)

# 'Close' data should be stationary before fitting ARIMA data['Close_diff'] =
data['Close'].diff().dropna()

# perform the ADF test on the original data
adf_result_original = adf_test(data['Close'])
p_value_original = adf_result_original.loc['p-value', 'value']
# check the p-value for stationarity and apply differencing if necessary
if p_value_original < 0.05:
        print('Data is already stationary.')
else:
        print('Data is not stationary. Differencing will be applied.')
```

```
        data['Close_diff'] = data['Close'].diff().bfill()  # Use backward fill to handle
any NaNs

        # Check stationarity again on differenced data
        adf_result_diff = adf_test(data['Close_diff'])
        p_value_diff = adf_result_diff.loc['p-value', 'value']
        if p_value_diff < 0.05:
                print('Differenced data is stationary.')
        else:
        print('Differenced data is still not stationary. Further investigation is
        required.')

# continue with data normalization, model loading, and other processing steps
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_data = scaler.fit_transform(data[['Close']])
# generate datasets for prediction
look_back = 60  # look-back period should match your model's training
configuration
X, y = create_dataset(scaled_data, look_back)

# split dataset into training and testing sets, ensure done before cleaning steps
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Now you can safely clean X_train and y_train
X_train = np.nan_to_num(X_train)
y_train = np.nan_to_num(y_train)

######################## ARIMA PREPROCESSING INC
############################
# generate datasets for prediction
look_back = 60  # should match your model's training configuration
X, _ = create_dataset(scaled_data, look_back)

# reshape data for the model
X = X.reshape((X.shape[0], X.shape[1], 1))

# load trained model
model = load_model('AAPL_CNN_with_attention.keras')
predictions = model.predict(X)

# inverse transform predictions
predictions = scaler.inverse_transform(predictions)
```

```
# plot results
plt.figure(figsize=(12, 6))
plt.plot(data.index[-len(predictions):], data['Close'][-len(predictions):],
label='Actual Stock Prices')
plt.plot(data.index[-len(predictions):], predictions.flatten(), label='Predicted Stock
Prices', alpha=0.7)
plt.title('Stock Price Predictions vs Actual Prices')
plt.xlabel('Date') plt.ylabel('Price')
plt.legend()
plt.show()

''' FIGURE 7 IMPLEMENTATION '''
residuals = data['Close'][-len(predictions):].values - predictions.flatten()

# plot residuals
plt.figure(figsize=(14, 7))
plt.subplot(1, 2, 1)
plt.plot(data.index[-len(residuals):], residuals, label='Residuals')
plt.title('Residuals')
plt.xlabel('Date')
plt.legend()

plt.subplot(1, 2, 2)
plt.hist(residuals, bins=50, density=True, label='Residuals Density', alpha=0.7)
plt.title('Density')
plt.xlabel('Residual Value')
plt.legend()

plt.tight_layout()
plt.show()

''' FIGURE 8 IMPLEMENTATION '''
# ensure 'Close' data is continuous and ordered for ARIMA fitting
arima_data = data['Close'].dropna().asfreq('B')  # 'B' for business day frequency
arima_data.ffill(inplace=True)  # Forward fill any missing values

# fit ARIMA model
arima_order = (5, 1, 0)  # adjust as necessary
arima_model = ARIMA(arima_data, order=arima_order)
arima_fit = arima_model.fit()

# get ARIMA residuals
arima_residuals = arima_fit.resid
```

```
# fit XGBoost on ARIMA residuals
xgb_model = XGBRegressor(n_estimators=100, objective='reg:squarederror')
xgb_model.fit(np.arange(len(arima_residuals)).reshape(-1, 1), arima_residuals)

# predict using XGBoost
xgb_predictions = xgb_model.predict(np.arange(len(arima_residuals)).reshape(-
1, 1))
# combine ARIMA model predictions and XGBoost predictions
combined_predictions = arima_fit.predict(start=arima_data.index[0],
end=arima_data.index[-1], typ='levels') + xgb_predictions

# plot actual and predicted values
plt.figure(figsize=(12, 6))
plt.plot(arima_data.index, arima_data, label='Actual Prices')
plt.plot(arima_data.index, combined_predictions[:len(arima_data)], label='ARIMA
+ XGBoost Predictions', alpha=0.7) plt.title('The ARIMA + XGBoost for Stock
Price Prediction') plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.show()

''' FIGURE 9 IMPLEMENTATION '''
# ensure 'Close' data is continuous and ordered for ARIMA fitting arima_data =
data['Close'].dropna().asfreq('B')  # 'B' for business day frequency
arima_data.ffill(inplace=True)  # forward fill missing values

# fit ARIMA model
arima_order = (5, 1, 0)  # example order, adjust as necessary
arima_model = ARIMA(arima_data, order=arima_order)
arima_fit = arima_model.fit()

# get ARIMA residuals
arima_residuals = arima_fit.resid
# predict residuals using XGBoost model
xgb_residual_predictions =
xgb_model.predict(np.arange(len(arima_residuals)).reshape(-1, 1))

# plot actual residuals and predicted residuals
plt.figure(figsize=(12, 6))
plt.plot(arima_residuals.index, arima_residuals, label='Residuals', alpha=0.7)
plt.plot(arima_residuals.index, xgb_residual_predictions, label='Predicted
Residuals', alpha=0.7, linestyle='--')
plt.title('ARIMA + XGBoost Residual Prediction')
```

```
plt.xlabel('Date') plt.ylabel('Residuals')
plt.legend()
plt.show()

''' FIGURE 10 IMPLEMENTATION'''
# load the LSTM model and history if saved, or ensure the LSTM training has
been run
lstm_model = load_model('lstm_model.h5')
history = pd.read_json('lstm_training_history.json')

# plot the training loss and validation loss
plt.figure(figsize=(10, 5))
plt.plot(history['loss'], label='Training Loss')
plt.plot(history['val_loss'], label='Validation Loss')
plt.title('LSTM: Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()

# assuming combined_predictions and arima_data cover same date range
evaluation_metric(arima_data.values, combined_predictions)

################### RESULTS SECTION ######################
# assuming get_*_predictions functions are defined and imported correctly
arima_actual, arima_pred = get_arima_predictions()
lstm_actual, lstm_pred = get_lstm_predictions()
xgboost_actual, xgboost_pred = get_xgboost_predictions()
cnn_actual, cnn_pred = get_cnn_attention_predictions()

# initialize results DataFrame
results = pd.DataFrame(columns=['Model', 'MSE', 'RMSE', 'MAE', 'R2'])
# create a list to hold data for the new rows
new_rows = []

# evaluate each model and append results to new_rows list
for model_name, actual, pred in [('ARIMA', arima_actual, arima_pred),
                                 ('LSTM', lstm_actual, lstm_pred),
                                 ('XGBoost', xgboost_actual, xgboost_pred),
                                 ('CNN Attention', cnn_actual, cnn_pred)]:
                        metrics = evaluation_metric(actual, pred)
                        new_rows.append({'Model': model_name, **metrics})

# convert new_rows to DataFrame and concatenate with results
```

```
new_rows_df = pd.DataFrame(new_rows)
results = pd.concat([results, new_rows_df], ignore_index=True)

plt.figure(figsize=(12, 2))
ax = plt.gca()
ax.axis('off')

# create table and display it
table = plt.table(cellText=results.values,
                                colLabels=results.columns,
                                loc='center',
                                cellLoc='center',
                                colColours=["#f2f2f2"] * results.shape[1])
table.auto_set_font_size(False)
table.set_fontsize(12)
table.scale(1.2, 1.2)
plt.title("Results of all Models", weight='bold', pad=15)

# save table as image file
plt.savefig("model_evaluation_results.png", bbox_inches='tight', dpi=300)

# optionally display the table on the screen
plt.show()
```

(6) utils_APPL.py Script

```
import numpy as np
import pandas as pd
from sklearn import metrics
from statsmodels.tsa.stattools
import adfuller
import statsmodels.api as sm # acf,pacf plot
import matplotlib.pyplot as plt

def adf_test(series):
        # p-value>0.562 or Critical Value(1%)>-3.44, non-stationary
        result = adfuller(series)
        output = pd.DataFrame(columns=['value'])
        output.loc['Test Statistic Value', 'value'] = result[0]
        output.loc['p-value', 'value'] = result[1]
        output.loc['Lags Used', 'value'] = result[2]
        output.loc['Number of Observations Used', 'value'] = result[3]
        output.loc['Critical Value(1%)', 'value'] = result[4]['1%']
        output.loc['Critical Value(5%)', 'value'] = result[4]['5%']
        output.loc['Critical Value(10%)', 'value'] = result[4]['10%']
        return output
def acf_pacf_plot(seq,acf_lags=20,pacf_lags=20):
        fig = plt.figure(figsize=(12, 8))
        ax1 = fig.add_subplot(211)
        fig = sm.graphics.tsa.plot_acf(seq, lags=acf_lags, ax=ax1)
        ax2 = fig.add_subplot(212)
        fig = sm.graphics.tsa.plot_pacf(seq, lags=pacf_lags, ax=ax2)
        plt.show()

def order_select_ic(training_data_diff):
        (p, q) = sm.tsa.arma_order_select_ic(training_data_diff, max_ar=6,
        max_ma=4, ic='bic')['bic_min_order']  # AIC
        print(p, q)  # 2 0

def order_select_search(training_set):
        df2 = training_set['close'].diff(1).dropna()
        pmax = 5 # pmax = int(len(df2) / 10)
        qmax = 5 # qmax = int(len(df2) / 10)
        bic_matrix = [] print('^', pmax, '^^', qmax)
        for p in range(pmax + 1):
                temp3 = []
                for q in range(qmax+1):
                        try:
                        # print('!', ARIMA(data['close'], order=(p, 1, q)).fit().bic)
```

```
                        # temp.append(ARIMA(data['close'], order=(p, 1, q)).fit().bic)
                        temp3.append(sm.tsa.ARIMA(training_set['close'], order=(p,
                        1, q)).fit().bic)
                        except:
                                temp3.append(None)
                bic_matrix.append(temp3)
        bic_matrix = pd.DataFrame(bic_matrix)
        # print('&', bic_matrix)
        # print('&&', bic_matrix.stack())
        # print('&&&', bic_matrix.stack().astype('float64'))
        p, q = bic_matrix.stack().astype('float64').idxmin()
        print('p and q: %s,%s' % (p, q))

def create_dataset(dataset, look_back=20):
        dataX, dataY = [], []
        for i in range(len(dataset)-look_back-1):
                a = dataset[i:(i+look_back)]  #don't use second dimension
                dataX.append(a)
                dataY.append(dataset[i + look_back])
                return np.array(dataX), np.array(dataY)



def evaluation_metric(y_test,y_hat):
        MSE = metrics.mean_squared_error(y_test, y_hat)
        RMSE = MSE**0.5
        MAE = metrics.mean_absolute_error(y_test,y_hat)
        R2 = metrics.r2_score(y_test,y_hat)
        return {'MSE': MSE, 'RMSE': RMSE, 'MAE': MAE, 'R2': R2}

def GetMAPE(y_hat, y_test):
        sum = np.mean(np.abs((y_hat - y_test) / y_test)) * 100
        return sum

def GetMAPE_Order(y_hat,y_test):
        zero_index = np.where(y_test == 0)
        y_hat = np.delete(y_hat, zero_index[0])
        y_test = np.delete(y_test, zero_index[0])
        sum = np.mean(np.abs((y_hat - y_test) / y_test)) * 100
        return sum

def NormalizeMult(data):
        data = np.array(data)
        normalize = np.arange(2*data.shape[1], dtype='float64')
```

```
    normalize = normalize.reshape(data.shape[1],2)
    print(normalize.shape)

    for i in range(0, data.shape[1]):
        list = data[:, i]
        listlow, listhigh = np.percentile(list, [0, 100])
        # print(i)
        normalize[i, 0] = listlow
        normalize[i, 1] = listhigh
        delta = listhigh - listlow
        if delta != 0:
            for j in range(0, data.shape[0]):
                data[j, i] = (data[j, i] - listlow)/delta
    # np.save("./normalize.npy",normalize)
    return data, normalize

def FNormalizeMult(data, normalize):
    #inverse NormalizeMult
    data = np.array(data)
    listlow = normalize[0]
    listhigh = normalize[1]
    delta = listhigh – listlow
    if delta != 0:
        for i in range(len(data)):
            data[i, 0] = data[i, 0] * delta + listlow
            return data
def NormalizeMultUseData(data,normalize):
    data = np.array(data)
    for i in range(0, data.shape[1]):
        listlow = normalize[i, 0]
        listhigh = normalize[i, 1]
        delta = listhigh – listlow
        if delta != 0:
            for j in range(0,data.shape[0]):
                data[j,i]  =  (data[j,i] - listlow)/delta
    return  data

def data_split(sequence, n_timestamp):
    X = []
    y = []
    for i in range(len(sequence)):
        end_ix = i + n_timestamp

        if end_ix > len(sequence) - 1:
```

```
            break

        seq_x, seq_y = sequence[i:end_ix], sequence[end_ix]
        X.append(seq_x)
        y.append(seq_y)
        return np.array(X), np.array(y)

def series_to_supervised(data, n_in=1, n_out=1, dropnan=True):
    n_vars = 1 if type(data) is list else data.shape[1]
    df = pd.DataFrame(data)
    cols, names = list(), list()
    # input sequence (t-n, ... t-1)
    for i in range(n_in, 0, -1):
        cols.append(df.shift(i))
        names += [('var%d(t-%d)' % (j + 1, i)) for j in range(n_vars)]
    # forecast sequence (t, t+1, ... t+n)
    for i in range(0, n_out):
        cols.append(df.shift(-i))
        if i == 0:
            names += [('var%d(t)' % (j + 1)) for j in range(n_vars)]
        else:
            names += [('var%d(t+%d)' % (j + 1, i)) for j in range(n_vars)]
        # put it all together
        agg = pd.concat(cols, axis=1)
        agg.columns = names


        # drop rows with NaN values
        if dropnan:
            agg.dropna(inplace=True)
            return agg

def prepare_data(series, n_test, n_in, n_out):
    values = series.values
    supervised_data = series_to_supervised(values, n_in, n_out)
    print('supervised_data', supervised_data)
    train, test = supervised_data.loc[:3499, :], supervised_data.loc[3500:, :]
    return train, test
```

REFERENCES

Abu-Mostafa, Y. S., & Atiya, A. F. (1996). Introduction to financial forecasting. *Applied intelligence*, 6, 205–213.

Alkhatib, K., Khazaleh, H., Ilkhanate, H.A., Alsoud, A.R., & Abualigah, L. (2022). A new stock price forecasting method using active deep learning approach. *Journal of Open Innovation: Technology, Market, and Complexity*, 8(2), 96. https://doi.org/10.3390/joitmc8020096

Bergstra, J., Bardenet, R., Bengio, Y., & K´egl, B. (2011). Algorithms for hyper-parameter optimization. *In Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011*, NIPS 2011 (p. 1–9).

Cahyadi, J., & Zahra, A. (2024). Bitcoin price prediction model development using convolutional neural network (CNN) and long short-term memory (LSTM). *Journal La Multiapp*, 5(2), 52-62. https://doi.org/10.37899/journallamultiapp.v5i2.1070

Eapen, J., Bein, D., & Verma, A. (2019). Novel deep learning model with CNN and bi-directional LSTM for improved stock market index prediction. In *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)* (pp. 0264–0270). IEEE. https://doi.org/10.1109/CCWC.2019.8666592

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems* (pp. 2672-2680).

Hanifi, S., Cammarono, A., & Zare-Behtash, H. (2023). Advanced hyperparameter optimization of deep learning models for wind power prediction. *Renewable Energy*, 221, 119700. https://doi.org/10.1016/j.renene.2023.119700

Haykin, S. (1999). *Neural networks: A comprehensive foundation* (2nd ed.). Prentice Hall.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8), 1735-1780.

Islam, M., Chen, G., & Jin, S. (2019). An overview of neural network. *American Journal of Neural Networks and Applications*, 5(1), 7-11. https://doi.org/10.11648/j.ajnna.20190501.12

Kim, K. J., & Han, we. (2000). Genetic algorithms approach to feature discretization in artificial neural networks for the prediction of stock price index. *Expert systems with Applications*, 19(2), 125-132.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, 521(7553), 436-444.

Liang, Y. (2021). Stock market forecasting based on artificial intelligence technology. *Electronic Theses, Projects, and Dissertations*. 1324. Retrieved from https://scholarworks.lib.csusb.edu/etd/1324``【oaicite:0】``.

Livieris, I. E., Kiriakidou, N., Stavroyiannis, S., & Pintelas, P. (2021). An advanced CNN-LSTM model for cryptocurrency forecasting. *Electronics*, 10(3), 287. https://doi.org/10.3390/electronics10030287

Masum, M., et al. (2021). Bayesian hyperparameter optimization for deep neural network-based network intrusion detection. In: *Proceedings of the 2021 IEEE International Conference on Big Data, Big Data 2021* (p. 5413–5419). IEEE. https://doi.org/10.1109/BigData52589.2021.9671576

Nasdaq. (2024). Historical stock prices and data for Apple Inc. *(AAPL)*. Retrieved 24 March 2024. https://www.nasdaq.com/market-activity/stocks/aapl/historical

O'Shea, K., & Nash, R. (2015). *An introduction to convolutional neural networks*. *arXiv preprint arXiv:1511.08458*.

Roll, R. (1992). Industrial Structure and the Comparative Behavior of International Stock Market Indices. *The Journal of Finance*, 47(1), 3–41. https://doi.org/10.2307/2329089

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

Shi, Z., Hu, Y., Mo, G., & Wu, J. (2022). Attention-based CNN-LSTM and XGBoost hybrid model for stock prediction. arXiv preprint arXiv:2204.02623. Retrieved from https://github.com/zshicode/Attention-CLX-stock-prediction.

Song, H., & Choi, H. (2023). Forecasting Stock Market Indices Using the Recurrent Neural Network Based Hybrid Models: CNN-LSTM, GRU-CNN, and Ensemble Models. *Applied Sciences*, 13(7), 4644. https://doi.org/10.3390/app13074644

Staffini, A. (2022). Stock price forecasting by a deep convolutional generative adversarial network. *Frontiers in Artificial Intelligence*, 5, 837596. https://doi.org/10.3389/frai.2022.837596

Tableau. (n.d.). *Time Series Forecasting: Definition, Applications, and Examples*. Retrieved from https://www.tableau.com/learn/articles/time-series-forecasting (Accessed February 10, 2024)

Van Houdt, G., Mosquera, C., & Nápoles, G. (2020). A review on the long short-term memory model. *The Artificial Intelligence Review*, 53(8), 5929–5955. https://doi.org/10.1007/s10462-020-09838-1

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems* (pp. 5998-6008).

Wu, J., Chen, X.-Y., Zhang, H., Xiong, L.-D., Lei, H., & Deng, S.-H. (2019). Hyperparameter Optimization for Machine Learning Models Based on

Bayesian Optimization. *电子科技学刊*, 17(1), 26–40. https://doi.org/10.11989/JEST.1674-862X.80904120

Yang, C.-H., & Chang, P.-Y. (2020). Forecasting the demand for container throughput using a mixed-precision neural architecture based on CNN–LSTM. *Mathematics*, 8(10), 1784. https://doi.org/10.3390/math8101784

Zhang, J., Ye, L., & Lai, Y. (2023). Stock price prediction using CNN-BiLSTM-Attention model. *Mathematics*, 11(9), 1985. https://doi.org/10.3390/math11091985

Zhu, R., Yang, Y., & Chen, J. (2023). XGBoost and CNN-LSTM hybrid model with Attention-based stock prediction. *IEEE 3rd International Conference on Electronic Technology*, Communication and Information (ICETCI), 359-365. doi:10.1109/ICETCI57876.2023.10176988

Zhu, T., Liao, Y., & Tao, Z. (2022). Predicting Google's Stock Price with LSTM Model. *Proceedings of Business and Economic Studies*, 5(5), 82-87.