Chapter 1

Introduction

1.1 Background and Motivation

It is very challenging to predict stock prices over time because of its intrinsic complexity and ambiguity over financial markets. Stock price fluctuations are affected by countless variables, including economic upsets, supply and demand, and even psychological behaviors of mood swings, which have a hard time being captured in quantifiable and predictive methods. Mathematical models seldom seem to capture the intricate nature of the market dynamics perfectly. Recurrent Neural Networks, especially Long Short-Term Memory (LSTM) networks, have proved promising to solve this problem by capturing effective temporal patterns of stock price data. Nevertheless, these models have drawbacks. One major disadvantage is the inherent lag in prediction where the forecasted values mostly lag behind the actual movement of the stock prices. This delay decreases the predictability of the model and its applicability in industries that demand sharp timing for making a profit in trading decisions.

In turn, by unifying advanced neural network architectures and preprocessing techniques, this dissertation could address the challenges mentioned, improve the accuracy and the timeliness of stock price forecasts, and integrate them into algorithmic trading, potentially becoming a competitive advantage for the latter over simple lagging indicators.

1.2 Objectives of the Study

The main objectives of this dissertation are:

1. To build up a more advanced stock price prediction model to be able to correctly forecast the next time step in real terms and by minimizing the lag between observed and predicted values.

2. To leverage RNNs and LSTM networks in conjunction with adaptive filtering and enhanced data preprocessing techniques to improve model performance.

3. To integrate the predictive model into algorithmic trading strategies, replacing or complementing lagging indicators like Moving Averages (MAs) and Relative Strength Index (RSI) with a forward-looking predictive system.

4. To maximize the profitability of trades by identifying optimal entry and exit points based on the predictive model's outputs.

1.3 Scope of the Dissertation

The scope of this dissertation includes the development, implementation, and evaluation of a stock price prediction framework using neural networks. The framework incorporates:

- Advanced deep learning architectures, specifically RNNs and LSTM networks.
- Adaptive preprocessing methods to improve the quality of input data.
- Integration with algorithmic trading strategies to test the practical usability of the predictive model in financial markets.

The focus is on daily stock price data, with an emphasis on predicting short-term price movements (e.g., the next day's closing price). While the framework is designed with stock markets in mind, the methodology can be extended to other financial instruments like commodities (e.g., oil prices) or indices.

1.4 Structure of the Dissertation

The dissertation is organized as follows:

- **Chapter 2: Literature Review**

  Explores the existing methods for stock price prediction, including traditional statistical approaches and advancements in neural network-based models. The chapter also highlights research gaps and positions the study within the context of existing literature.

- **Chapter 3: Frameworks and Models for Stock Price Prediction**

  Describes the proposed architectures and models in detail. This includes hybrid architectures such as the LSTM-RLS model, the DNS architecture, the ARIMA-LSTM Residual Integration Framework, and the Multi-Feature LSTM Forecasting Framework. The chapter also discusses data preprocessing techniques, training methodologies, and feature engineering processes.

- **Chapter 4: Results and Evaluation**

  Presents the experimental results and evaluates the performance of the proposed models using various metrics. Visualization of results and detailed analysis are included for each framework, highlighting their predictive capabilities and limitations.

- **Chapter 5: Discussions and Challenges**

  Summarizes the key findings, addresses the challenges encountered during the model development process, and discusses the implications of the results. This chapter also explores potential refinements to improve the models further.

- **Chapter 6: Conclusion and Future Direction**

  Concludes the dissertation by summarizing the key contributions and insights gained from the study. Additionally, it outlines potential directions for future research and model enhancements.

- **Chapter 7: Publications**

  Lists the publications resulting from the work presented in this dissertation.

Chapter 2

Literature Review

2.1 Overview of Stock Price Prediction Methods

Stock price prediction has been a subject of extensive research, employing both traditional statistical methods and modern machine learning techniques. Traditional approaches, such as ARIMA models, rely heavily on the detection of linear patterns in historical data and therefore are not good enough to capture complex and nonlinear dynamics in financial markets. Recent advances in machine learning have introduced neural networks, particularly Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks, as promising alternatives. These models are very good at capturing temporal dependencies and nonlinear relationships, making them very suitable for time-series forecasting.

To overcome the limitations of single-model approaches, researchers have increasingly explored hybrid architectures that integrate deep learning models with advanced preprocessing techniques like Kalman Filters and Wavelet Transforms. These combinations enhance prediction accuracy by reducing noise and captur-

ing complex patterns in financial data. A key consideration that emerges in these architectures is the tradeoff between reducing prediction lag and minimizing prediction loss. Some architectures, such as DNS and ARIMAX-LSTM residual models, focus on reducing lag to improve responsiveness, while others, including pure LSTM and GAN-based models, are optimized to reduce prediction error but often exhibit temporal lag.

2.2 RNNs and LSTMs in Time-Series Forecasting

Recurrent Neural Networks (RNNs) and their advanced variants, LSTMs, have demonstrated exceptional capabilities in time-series forecasting. LSTMs, with their ability to capture long-term dependencies, address the vanishing gradient problem that limits traditional RNNs.

For instance, Dastgerdi and Mercorelli [1] introduced an LSTM-based approach integrated with Kalman Filters and Wavelet Transforms for financial forecasting. Wavelet Transforms were used for signal decomposition, enhancing the signal-to-noise ratio, while the Kalman Filter refined the LSTM's predictions by filtering out noise. This synergy improved accuracy and reliability over traditional methods.

Similarly, Fang, Cai, Fan, *et al.* [2] proposed a Kalman-LSTM model for short-term traffic flow prediction, showcasing how preprocessing with Kalman Filters can improve the quality of inputs for LSTMs. These studies highlight the importance of combining noise reduction techniques with the temporal modeling capabilities of LSTMs. However, such models, although accurate, tend to introduce a slight lag in the predictions due to their dependency on prior temporal

sequences.

2.3 Applications of Hybrid Architectures in Time Series Forecasting

Hybrid models combining traditional statistical methods, preprocessing filters, and deep learning networks have emerged as powerful tools for time series forecasting.

Song, Li, Cheng, *et al.* [3] proposed a Kalman Filter Fusing LSTM (KF-FLSTM) model for tracking nonlinear dynamics, addressing challenges in environments with non-Gaussian noise. The model combined LSTMs to analyze complex nonlinear patterns, but the predictions were optimized in the light of historical information by applying Kalman filters.

In another study, Tian, Lian, Wang, *et al.* [4] used a KF-LSTM algorithm in ultra-wideband indoor positioning systems, which proved its application in noise filtering and location estimation with robust accuracy. Similarly, Wang, Li, Cao, *et al.* [5] combined Adaptive Extended Kalman Filters with LSTM networks for the states of charge of lithium-ion batteries, where LSTMs were used to capture the nonlinear patterns and enhance prediction accuracy.

In the stock market domain, Shah, Bhatt, and Shah [6] introduced a hybrid ARIMA-LSTM model. The ARIMA model predicted linear trends, while the LSTM captured residual nonlinearities, resulting in improved accuracy on Indian stock market data. These hybrid approaches underscore the potential of combining statistical and deep learning methods for enhanced forecasting. Notably, the

ARIMA-LSTM architecture is one of the few that strikes a balance between reducing error and minimizing lag by separating trend and residual learning.

2.4 GANs in Time-Series Forecasting

Generative Adversarial Networks (GANs) have emerged as powerful tools for modeling complex data distributions, particularly in image and text generation. Recently, their application has expanded into the time-series domain to address challenges in capturing the stochastic nature of sequential data.

Traditional supervised learning models used in time-series forecasting tend to be deterministic, which limits their ability to simulate the intrinsic randomness of financial markets. In contrast, GANs generate new samples by learning to mimic the underlying distribution of the training data, offering more flexible generative capabilities.

Yoon and Jarrett [7] proposed Time-series GANs (TimeGAN), a hybrid architecture that combines adversarial training with supervised learning in a joint embedding space. TimeGAN ensures that the generated sequences preserve temporal dependencies and variable correlations by integrating both reconstruction loss and adversarial loss. This dual-objective approach significantly improves the realism and predictive power of generated sequences compared to conventional GANs.

Building on the generative capability of GANs, Gu, Du, and Wang [8] introduced the RAGIC (Risk-Aware Generative Framework for Stock Interval Construction) architecture. Unlike typical point-prediction models, RAGIC focuses on generating realistic price intervals that account for market uncertainty and risk.

Its generator learns both global and local market trends, simulating plausible future price paths infused with market randomness. By performing statistical inference on these generated sequences, RAGIC constructs dynamic prediction intervals with adaptive widths that reflect market volatility.

A key innovation in RAGIC lies in its risk-aware interval estimation, which maintains consistent coverage (95%) while keeping the interval width narrow. This is particularly useful for decision-making in financial domains, where understanding the range of possible outcomes is often more informative than single-point forecasts.

While GAN-based methods such as TimeGAN and RAGIC significantly reduce prediction loss and better capture stochastic market behavior, they often exhibit a lag in real-time scenarios due to their generative sampling nature. Thus, they are highly suitable for long-term scenario generation and interval forecasting but may not be ideal for high-frequency prediction tasks requiring immediate responsiveness.

2.5 Research Gaps and Positioning of the Study

Despite significant advancements, several challenges remain in stock price prediction:

- **Tradeoff Between Prediction Lag and Prediction Loss:** Some models focus on minimizing the time delay in capturing real-time price movements (lag), such as the Dual Network Solution (DNS) and ARIMAX-LSTM residual architectures [9]. These models are beneficial for real-time trading applications. On the other hand, architectures like LSTM, TimeGAN, and

RAGIC are optimized to reduce prediction loss and capture uncertainty but may produce lagged responses.

- **Integration of Adaptive Filters with Neural Networks:** Most existing research focuses on using Kalman Filters and similar techniques for data preprocessing, rather than integrating them into the neural network's weight update mechanism. This presents an opportunity to explore real-time learning models that combine the adaptability of recursive filters with the pattern recognition capabilities of LSTMs.

- **Capturing Deep Nonlinearities:** Current hybrids (e.g., ARIMA-LSTM, Kalman-LSTM) improve over single models but still rely on relatively shallow nonlinear transformations. There is a need for architectures that explicitly learn higher-order interactions—such as attention-based layers, convolutional preprocessing, or adversarial generators—to better model the complex, nonstationary dynamics of financial time series.

- **Lack of Movement Prediction Metrics:** Traditional evaluation metrics like Mean Squared Error (MSE) fail to capture the effect of lag properly in real-time prediction. Novel metrics, including the Movement Prediction Metric recently proposed by Samanta, Pratama, Sundaram, *et al.* [9], provide better insights into model performance beyond error magnitude.

This dissertation aims to address these gaps by developing a novel hybrid framework that integrates recursive filtering techniques with LSTMs, focusing on reducing prediction lag while retaining the accuracy and risk-awareness advantages of deep generative models. By evaluating models not only in terms of

error but also responsiveness, this study seeks to provide practical solutions for real-time financial forecasting.

Chapter 3

Model Architectures for Stock Price Prediction

3.1 Overview of Proposed Architectures

This chapter presents the various architectures and frameworks explored for stock price prediction, from initial experiments during the summer internship to advanced hybrid and feature-rich frameworks developed later. The focus is on improving prediction accuracy, addressing issues like lag, and incorporating domain-specific features. A systematic approach to model building, including feature engineering and model integration, is discussed.

3.2 Hybrid LSTM-RLS Architecture

3.2.1 Workflow and Design

The Hybrid LSTM-RLS Architecture was the first model developed during my summer internship. It combines the strengths of LSTM for capturing sequential dependencies and Recursive Least Squares (RLS) for refining predictions. Figure 3.1 illustrates the architecture, highlighting the key components and their interactions.

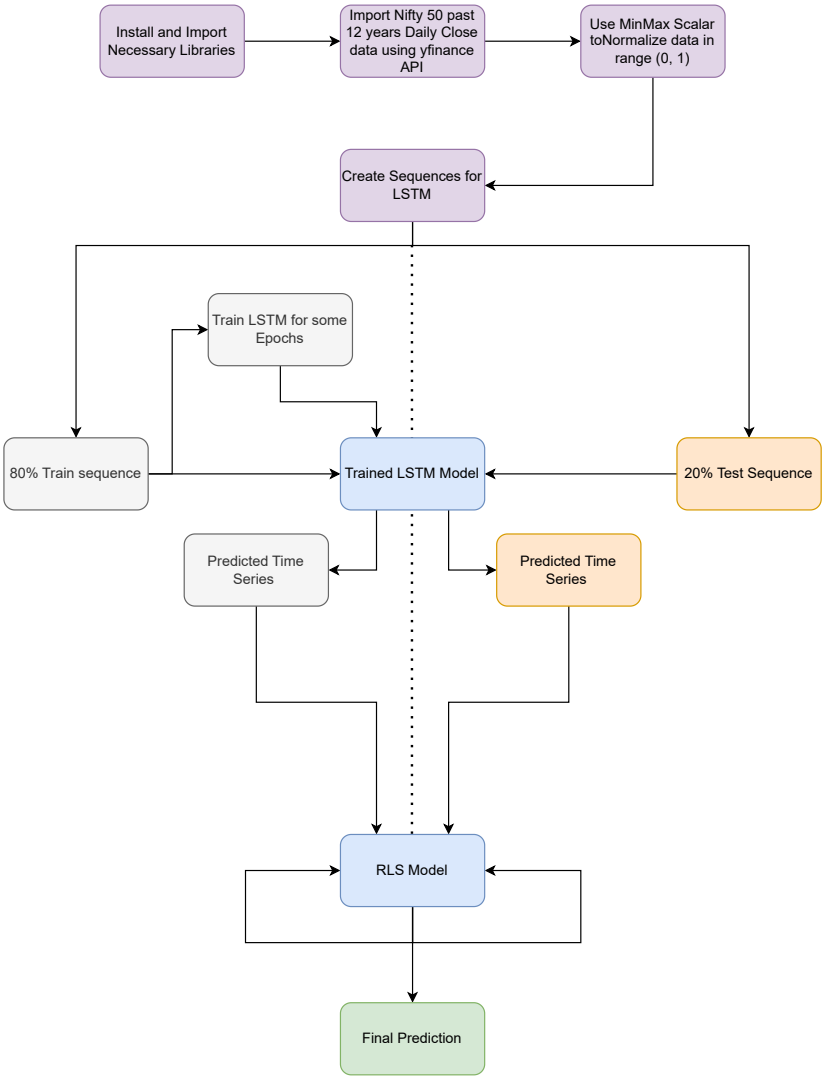

Figure 3.1: Hybrid LSTM-RLS Architecture

The workflow for this architecture is as follows:

1. **Install and import libraries:** TensorFlow, NumPy, yfinance, and scikit-learn.

2. **Fetch stock data:** Historical prices retrieved via yfinance API.

3. **Data preprocessing:** The data was normalized, missing values were handled, and it was split into training and testing sets.

4. **Create sequences:** Price sequences of fixed length were created as input to the LSTM model.

5. **Train the LSTM model:** The LSTM model was trained to predict the next stock price based on these sequences.

6. **LSTM predictions passed to RLS model:** Scalar predictions from the LSTM model were passed to the RLS model.

7. **RLS model provides final prediction:** The RLS model used the LSTM predictions to produce refined stock price predictions.

3.3 Enhanced Hybrid LSTM-RLS Architecture

To address issues such as dimension mismatches and long training times in the original architecture, several enhancements were implemented during Semester 3. These refinements simplified the process and improved predictive accuracy.

Figure 3.2 illustrates the improved workflow. The modifications made the architecture more efficient and straightforward for stock price forecasting.

3.3.1 Architecture Workflow

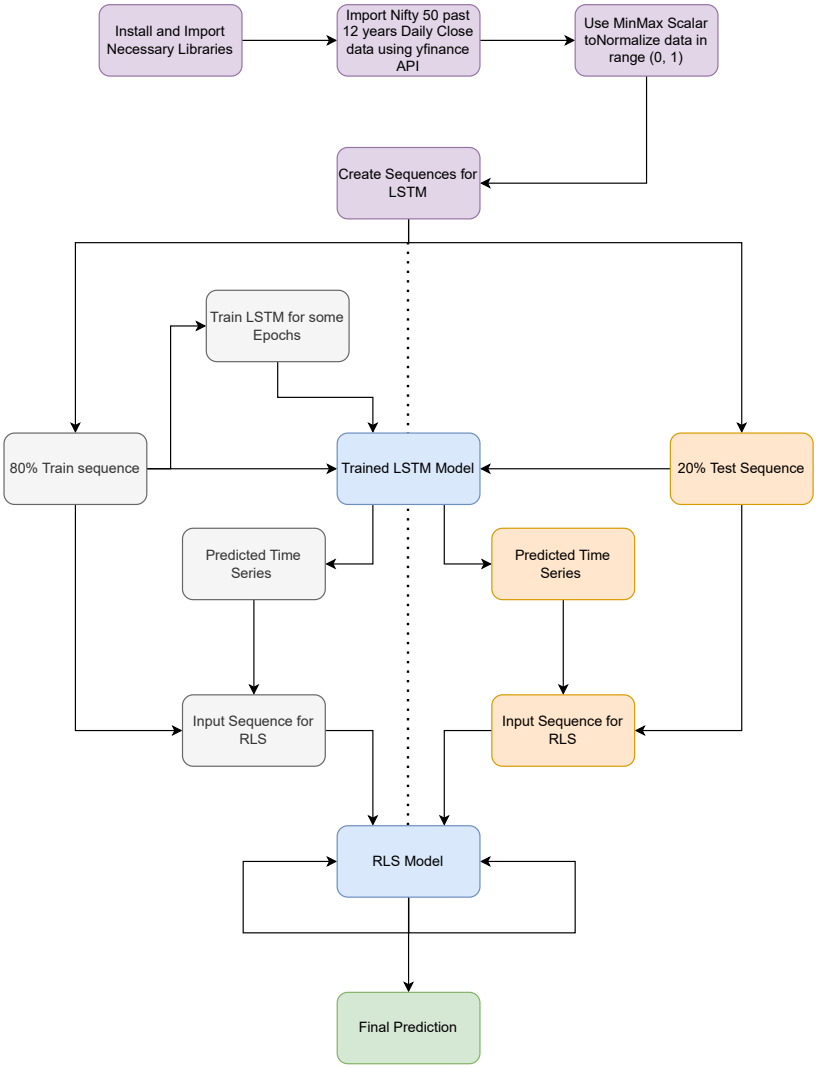

Figure 3.2: Enhanced Hybrid LSTM-RLS Architecture

The architecture process involves the following steps:

1. **Install and import necessary libraries:**

   - Libraries such as `yfinance`, `tensorflow`, `scikit-learn`, and others were imported.

2. **Import Nifty 50 daily close price data:**

    - Closing prices of the Nifty 50 index for the last 12 years were fetched daily using the `yfinance` API.

3. **Normalize the data:**

    - Data normalized to [0, 1] using `MinMaxScaler`.

4. **Sequence creation for LSTM:**

    - Sequences of historical stock prices were created as inputs for training the LSTM model.

5. **Split dataset into training and testing set**

    - The sequences were divided into training (80%) and testing (20%) sets to evaluate model performance.

6. **Training of the LSTM model:**

    - The LSTM model was trained on the 80% training data over several epochs to predict stock prices.

7. **Input Sequences creation for RLS:**

    - Predictions from the LSTM model is used to prepare input sequences for the RLS model.

8. **Forecast Stock Prices with RLS:**

    - The RLS model processed the LSTM predictions as inputs, producing the final stock price forecasts.

3.3.2 Developing Input Sequences for RLS

Figure 3.3 provides a detailed view of the sequence development process for the RLS model based on LSTM predictions.

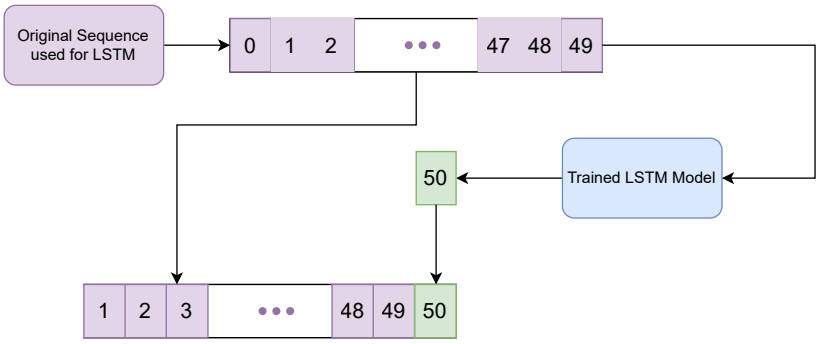

Figure 3.3: Input Sequence Formation for RLS

The following steps outline the process:

1. **Original Sequence for LSTM:**
    - The original sequence of stock prices served as the input for training the LSTM model.
    - For example, a sequence ranging from index 0 to 49 was selected.
2. **Training the LSTM Model:**
    - The LSTM model was trained on the input sequence.
    - Upon training, the model predicted the 50th value based on the prior 49 values.
3. **Shifting the Sequence for RLS Input:**
    - The sequence was shifted by one step to generate input for the RLS model.

- For instance, the updated sequence started at index 1 and extended to index 50, incorporating the LSTM prediction for the 50th point.

4. **Final Input for RLS Model:**

   - The shifted sequence (from index 1 to 50) was provided as input to the RLS model.
   - The RLS model used this sequence to produce the final stock price prediction.

This sliding window approach facilitated seamless integration between the LSTM and RLS models, ensuring improved accuracy and efficient stock price forecasting.

3.3.3 Multi-Feature Hybrid LSTM-RLS Implementation

The preprocessed dataset was directly used in the **Hybrid LSTM-RLS architecture**. See **Figure 3.4** for the workflow.

The LSTM model was trained following the standard sequence based approach, where multi-feature input sequences were constructed and used for training.

Key Modification: RLS Input Sequence Construction

A significant deviation from the previous approach was the method used to construct input sequences for the **Recursive Least Squares (RLS) model**. As illustrated in **Figure 3.5**, the input sequence creation for RLS followed a structured pipeline:

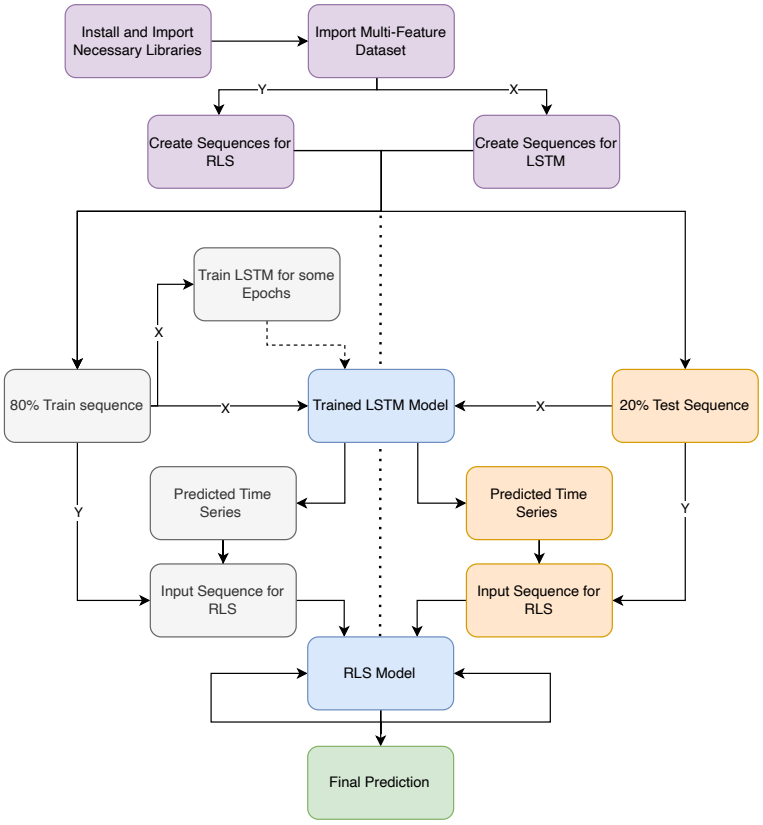

Figure 3.4: Flow diagram of Multi-Feature Hybrid LSTM-RLS implementation.

- Instead of deriving the RLS input sequences from the **LSTM input sequence**, the sequences were generated using **returns and close prices**.
- The LSTM model was trained on sequences created from the **multi-feature input** (**X**). Once trained, the predicted values from LSTM were appended to the sequences constructed from **daily returns** (**Y**), ensuring that the RLS model receives inputs of the same nature as its output, thus allowing effective prediction in an online setting.
- Two separate training strategies were employed:

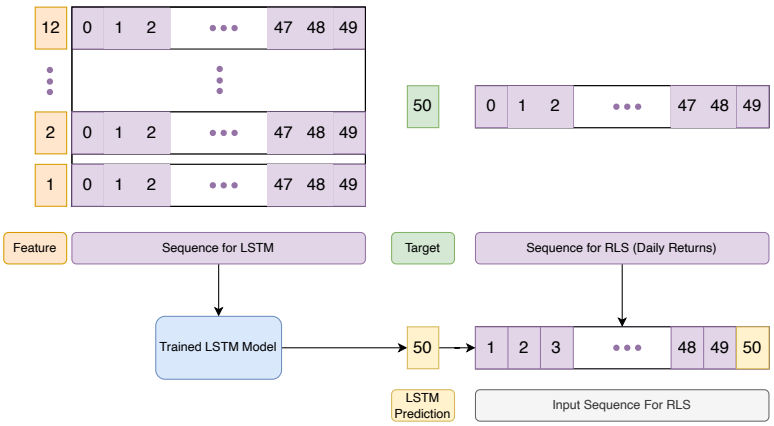

Figure 3.5: RLS input sequence construction using LSTM outputs.

- **Predicting Returns**: The model was trained to forecast stock returns, leveraging the structured relationship between past returns and future movements.
- **Predicting Close Prices**: The model was also trained using close prices to evaluate its performance in direct price prediction.

Impact of this Approach

- **Alternative Feature Representation**: By constructing input sequences from returns and close prices, the model captured different aspects of market behavior.
- **Comparative Performance Analysis**: Training for both returns and close prices allowed an in-depth evaluation of prediction effectiveness under different target variable choices.
- **Flexible RLS Adaptation**: The RLS model benefited from this new input structure, enabling it to adjust dynamically to either return-based or price-

based forecasts.

This modification provided insights into how input sequence construction affects the overall model accuracy and adaptability, further refining the **Hybrid LSTM-RLS architecture**.

3.4 Dual Network Solution (DNS) Architecture

3.4.1 Design and Integration

The Dual Network Solution (DNS) architecture addresses the one-time-step lag observed in the previous models. Figure 3.6 shows the DNS architecture, which involves two networks: one trained on trends and the other on actual data.

Steps include:

1. **Trend extraction:** Network 1 is trained to predict trends using slope differences or moving averages.
2. **Prediction refinement:** Network 2 uses the output of Network 1 to generate lag-free predictions.

3.5 Advanced Data Preprocessing for Enhanced Model Performance

3.5.1 Challenges in Scaling and Activation

During the development of the stock price prediction models, challenges arose when scaling the data using traditional normalization techniques such as Min-

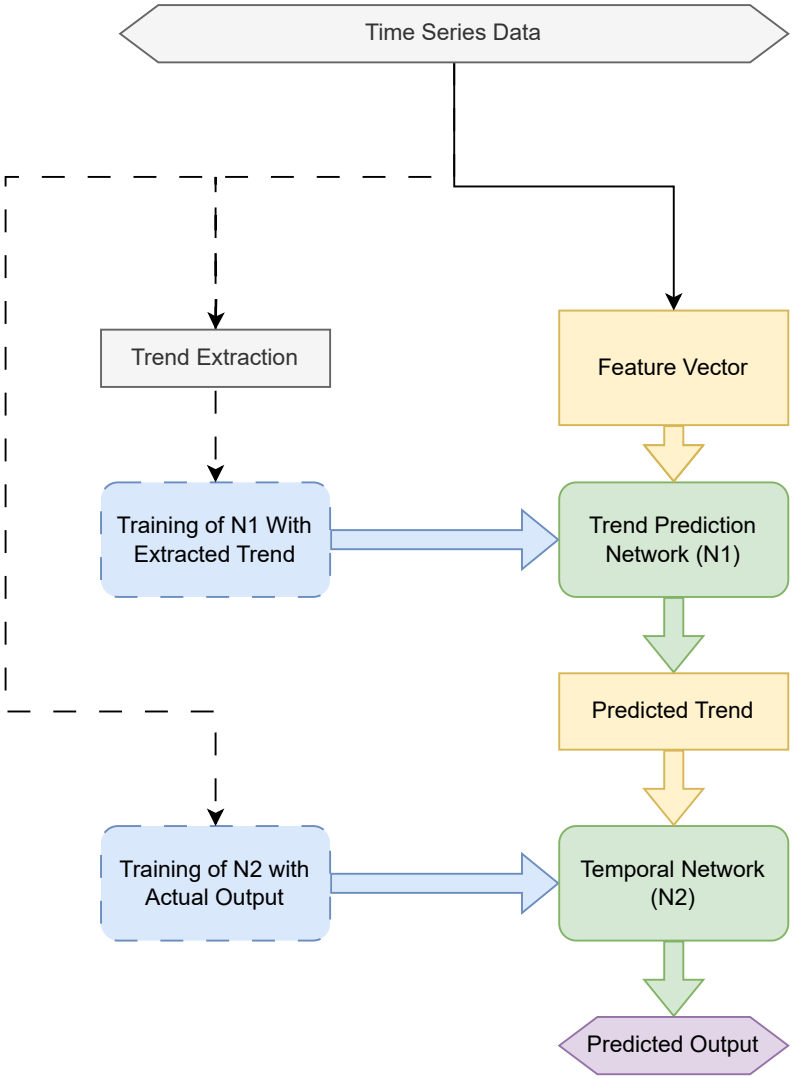

Figure 3.6: DNS Architecture

MaxScaler. This approach limits the range of predictions to the maximum value observed during training, potentially capping future predictions. To address this issue, an alternative preprocessing technique was introduced, which involves:

- **Log Transformation:** Applying a logarithmic transformation to the close price values to reduce large variations and stabilize the data.

- **Standardization:** Standardizing the log-transformed values to ensure that the data is centered and scaled, allowing it to be processed effectively by the model.

After preprocessing, the resulting data ranged approximately between -2 and 2. To handle this range, the `tanh` activation function was selected for its natural prediction range of $[-1, 1]$. However, to ensure flexibility in predictions beyond this range:

- A multiplier of 2.5 was applied after the `tanh` activation function. This adjustment extended the prediction range to $[-2.5, 2.5]$, providing additional room for extreme values without capping predictions.

Figure 3.7 shows the distribution plot of the raw close values, log-transformed close values, and scaled log-transformed values. This plot depicts how the preprocessing steps have allowed the data to fall in a range that the activation function can handle effectively.

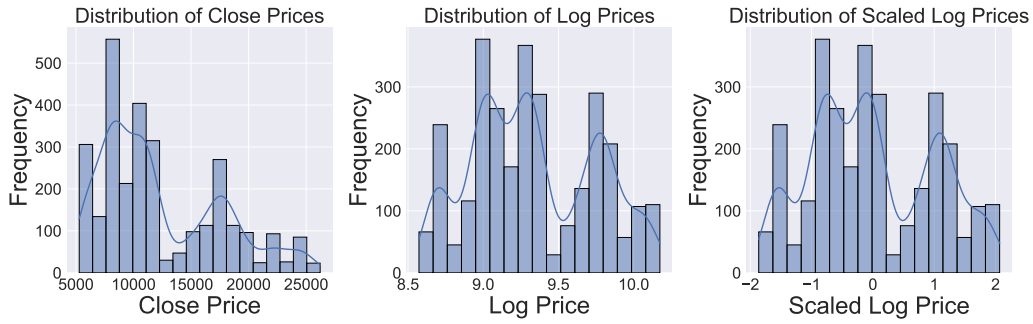

Figure 3.7: Distribution plot of raw close values, log-transformed close values, and scaled log-transformed close values.

3.5.2 Application in DNS Architecture

This preprocessing technique, as explained in Section 3.5.1 was used in the concept of the Dual Network Solution Architecture. Using log transformation, standardization, and modified range of activation functions, the DNS model represents both linear and nonlinear parts of stock price movements.

This preprocess step was crucial in enhancing the prediction accuracy of the model. racy of the DNS architecture, so that it could support a greater number of values While the predictions of strength were retained, the changes had provided the basis for groundwork for subsequent enhancements in model design and integration.

3.6 ARIMAX-LSTM Residual Integration Framework

3.6.1 ARIMAX-LSTM Hybridization

The initial approach involved integrating the capability of ARIMA to model linear patterns and LSTM's strength in capturing non-linear relationships. However, due to suboptimal performance, the ARIMA model was replaced with ARIMAX for handling the linear component while LSTM continued to process residuals. The final prediction was obtained by integrating the outputs of these two models. The revised framework consists of:

- **ARIMAX model for linear trends:** Unlike ARIMA, ARIMAX incorporates external features alongside past values to better capture the linear com-

ponent of stock prices.

- **LSTM model for residuals:** LSTM models the remaining non-linear residuals after ARIMAX processing.
- **Final integration:** Predictions from both models are combined to enhance accuracy.

3.6.2 Challenges and Model Refinement

Initially, the ARIMA-LSTM hybrid model was tested; however, its performance was unsatisfactory. The residuals did not capture sufficient differences between the original close price values and ARIMA's linear predictions, leading to subpar results.

To address this issue:

- The **ARIMA model was replaced with ARIMAX**, incorporating external variables to improve the modeling of linear relationships.
- The **LSTM model remained responsible for capturing non-linear residuals**.
- The integration of **ARIMAX and LSTM** was evaluated to determine if this modification led to improved performance.

This refined framework aimed to leverage ARIMAX's enhanced linear modeling alongside LSTM's capacity for non-linearity, ultimately improving the predictive capability of the model.

3.6.3 Applying RLS to Residuals Framework Output

In the **ARIMAX-LSTM-RLS model**, after predicting returns using the **ARIMA-LSTM Residual Integration Framework**, the next step involved incorporating **Recursive Least Squares (RLS)** to refine the final predictions.

RLS Input Sequence Construction

To enhance the prediction accuracy, input sequences for RLS were constructed as follows:

- The predicted returns obtained from the **residuals architecture** were used to generate input sequences for RLS.
- RLS was then applied to these sequences, providing the **final predicted returns output**.

Impact of RLS on the Residuals Architecture

- **Sequential Refinement**: By feeding the residuals architecture's output into RLS, the model aimed to further refine the predictions.
- **Performance Evaluation**: The final output from RLS was analyzed to determine whether this additional step improved the overall prediction accuracy.

This integration of RLS with the residuals architecture provided valuable insights into whether the additional layer of processing contributed to enhanced model performance.

3.7 Multi-Feature LSTM Forecasting Framework

3.7.1 Overview and Objectives

The Multi-Feature LSTM Forecasting Framework aims to enhance stock price prediction accuracy by incorporating a diverse set of features derived from technical indicators and price patterns. These features include daily returns, Relative Strength Index (RSI), On-Balance Volume (OBV), and various ratios capturing short-term and long-term price movements.

The key objectives of this framework are:

- Many key features for an integrated understanding of Market Behavior.
- Developing a methodology of training that is robust and adaptive for daily usage.
- It uses the strength of LSTM in capturing temporal dependency. cies within financial data.

The framework is designed based on the model's adaptability with respect to newer ideas. Market trend with scalability and computational effectiveness of a solution. Incorporation of feature selection and refinement adds further sophistication to the predictivity ability of the model.

3.7.2 Training Methodology

This is specific for the Multi-Feature LSTM Forecasting Framework. Designed for high-level real-time stock market information processing and daily changes. The methodology involved the following:

One-Step Prediction and Incremental Training

That makes the prediction of the stock price at the next step a function of only Minimize the error between the predicted and actual values for the immediate next day. Even though it is predicting several future time steps, only the first The predicted value is used in the context of algorithmic trading for decision making. Immediately after obtaining tomorrow's true value, the training and testing datasets are updated:

1. Function that omits the first value in the training dataset and the last value from the test dataset is appended to keep the no of training sample sizes.

2. The first value in the test data set is removed, and the newly real value is returned by the final entry of available.

This means the model is updated on the most recently available data points. It enables it to adjust to changes in the market. The model is trained daily With such an updated dataset, predictions of the following day can be made

Visualization of Incremental Training Process

The incremental training process is visualized in Figure 3.8. This figure illustrates the dynamic updates to the training and testing datasets as new data becomes available. The real-time adaptability of the model is a key feature that supports its application in live trading environments.

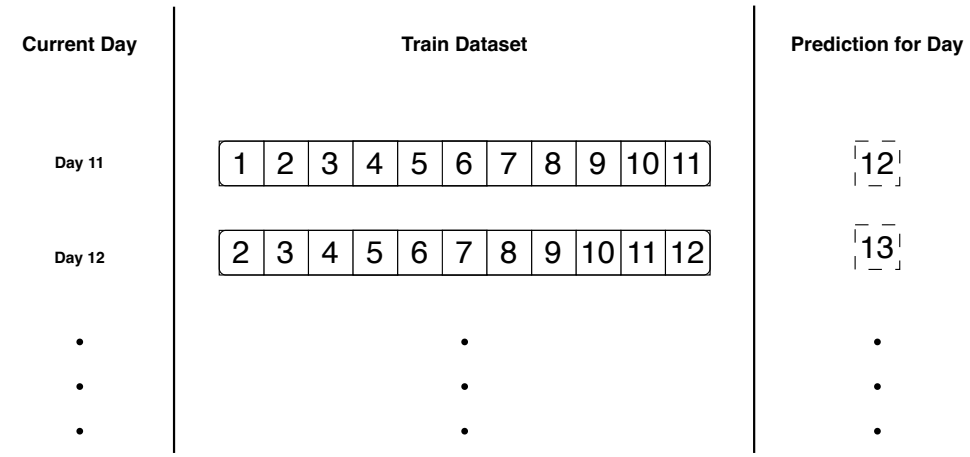

Figure 3.8: Visualization of the Incremental Training Process for Multi-Feature LSTM Forecasting Framework.

3.7.3 Feature Engineering Techniques

This framework incorporates multiple features, such as returns, RSI, and OBV, to enhance predictive accuracy. All features were normalized for consistency.

The distribution of all features in the dataset is shown in Figure 3.9. This plot highlights the range and variability of each feature, demonstrating the effectiveness of normalization.

Additionally, the correlation matrix for all features is displayed in Figure 3.10. This plot provides insights into feature interdependencies and the strength of their relationships with the target variable.

Finally, Figure 3.11 presents the seasonal decomposition plot of NSEI close values, illustrating the observed trend, seasonality, and residual components. This visualization assists in understanding the underlying temporal patterns of the target variable.

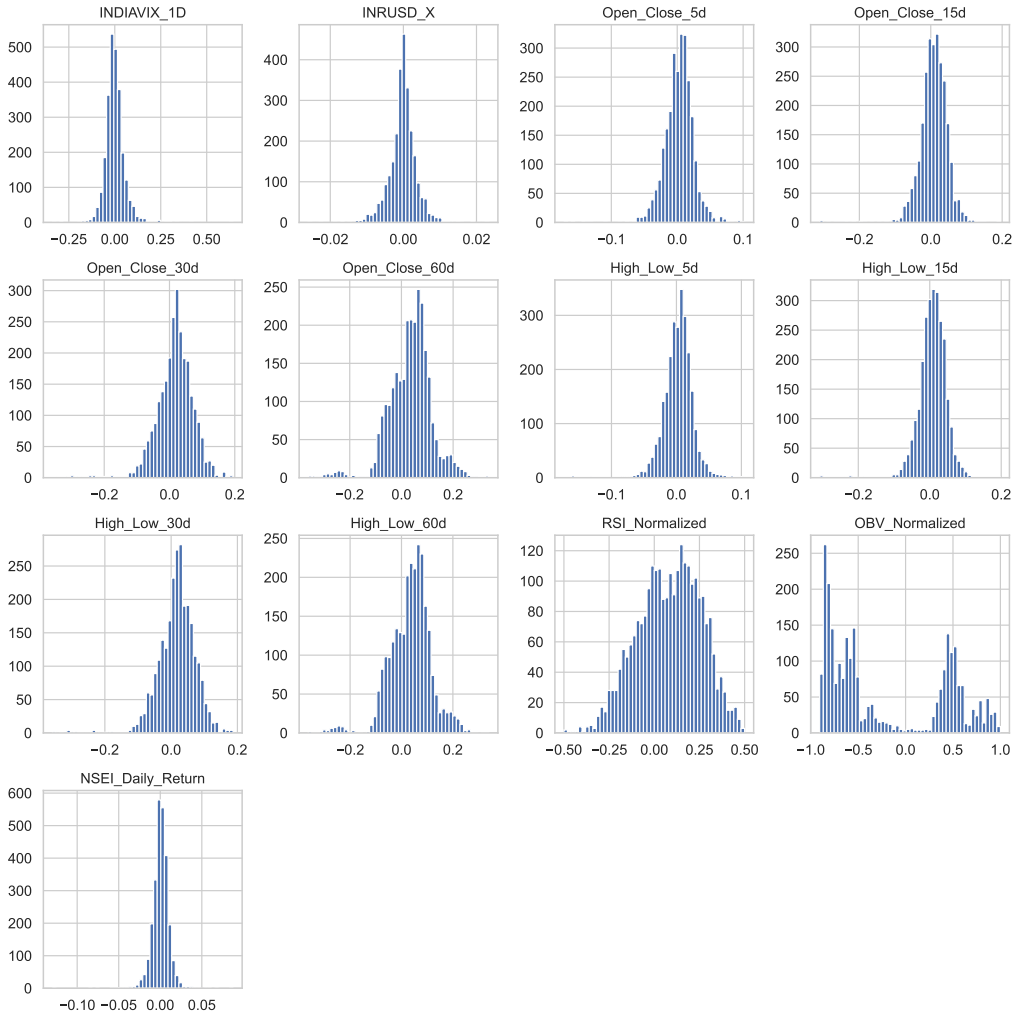

Figure 3.9: Distribution plots of all features used in the forecasting framework.

3.7.4 Feature Selection Using SelectKBest

Feature selection was performed using the SelectKBest method with f_regression.

The feature importance scores are presented in Table 3.1.

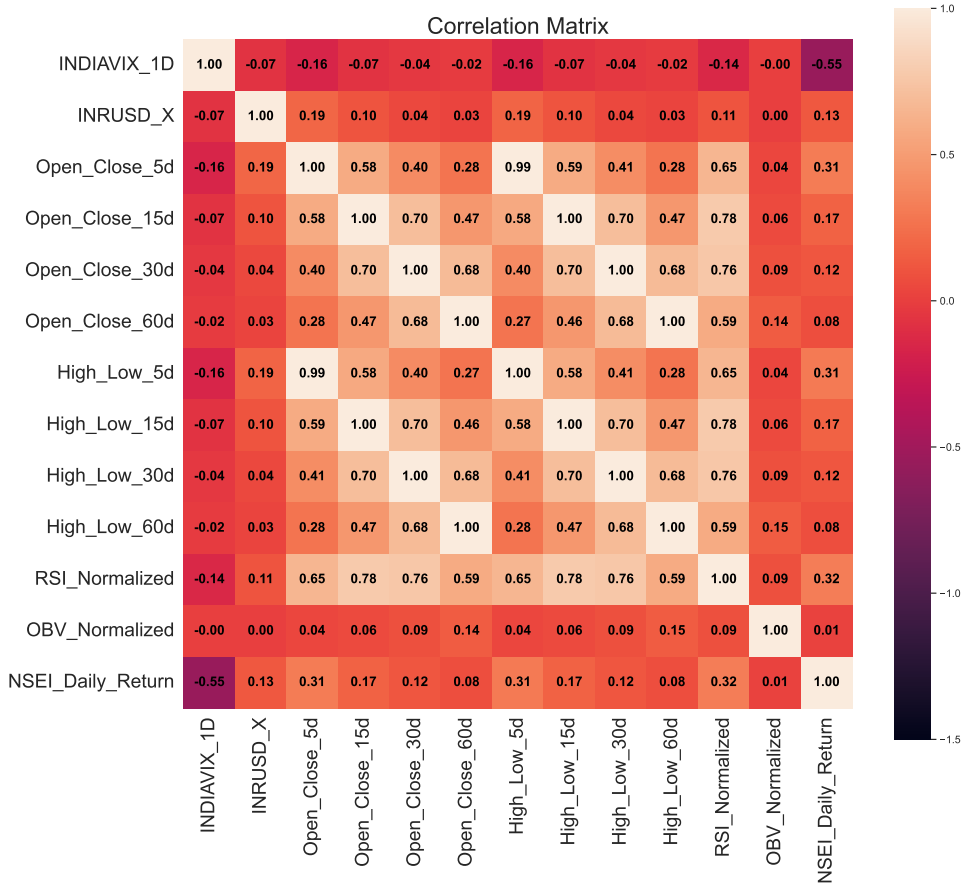

Figure 3.10: Correlation matrix of all features in the dataset.

3.7.5 Feature Refinement

Low-importance features such as OBV_Normalized and 30-day/60-day returns were dropped. The refined feature set ensured better model performance with reduced complexity.

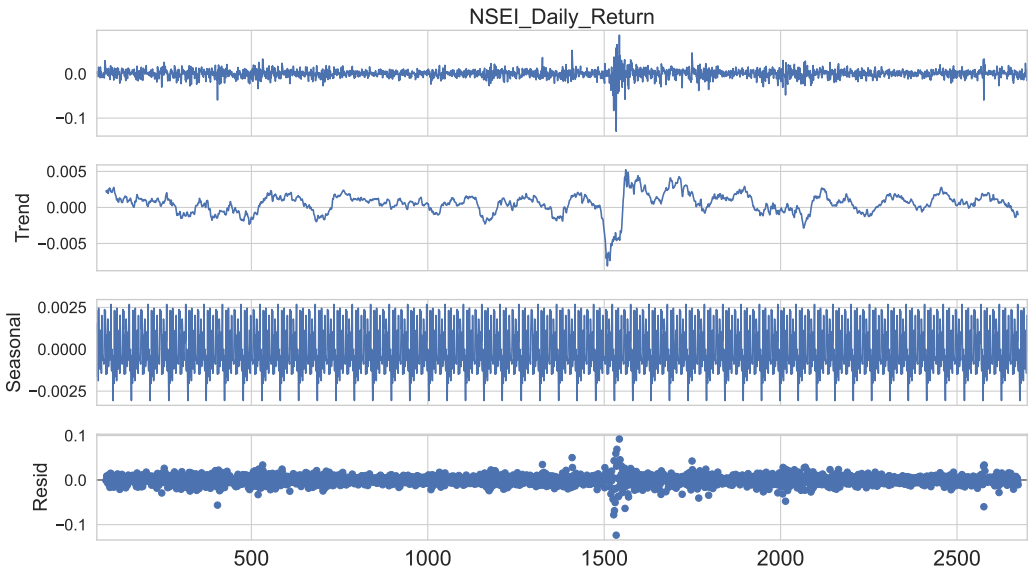

Figure 3.11: Seasonal decomposition of NSEI close values.

Table 3.1: Feature Importance Scores using SelectKBest

Feature	**Score**
INDIAVIX_1D	1125.250181
RSI_Normalized	292.775295
Open_Close_5d	276.984547
High_Low_5d	268.947328
High_Low_15d	79.876745
Open_Close_15d	79.634278
INRUSD_X	42.357608
Open_Close_30d	35.719346
High_Low_30d	35.259659
Open_Close_60d	16.824506
High_Low_60d	16.235917
OBV_Normalized	0.570088

3.8 GARCH for Stock Price Prediction

3.8.1 Introduction to GARCH

The Generalized Autoregressive Conditional Heteroskedasticity (GARCH) model is widely used in financial time series modeling due to its ability to capture volatility clustering. In stock markets, volatility tends to persist over time, meaning that periods of high volatility are often followed by more volatile periods, and similarly for low volatility. Since the target variable in this study is **daily returns**, which inherently exhibits volatility characteristics, GARCH is a suitable choice for modeling this behavior.

GARCH is particularly relevant to this study because it models conditional variance, providing insights into future uncertainty levels. By leveraging GARCH, we aim to enhance stock return prediction by explicitly modeling the variance dynamics, which traditional models might overlook.

3.8.2 Model Implementation

To implement GARCH, the following approach was used:

- **Mean Prediction using ARIMAX:** Before applying GARCH, the mean component of stock returns was modeled using the ARIMAX model. ARIMAX extends ARIMA by incorporating external regressors, making it suitable for capturing linear dependencies in stock returns.

- **Residuals Extraction:** After obtaining the ARIMAX predictions, the residuals (unexplained variations) were extracted. These residuals capture the unpredictable fluctuations in returns.

- **GARCH Modeling on Residuals:** The extracted residuals were then used

to train a GARCH model, which estimated the conditional variance of the returns. The GARCH model accounted for volatility clustering, ensuring that periods of high variance were correctly modeled.

3.8.3 Integration with Existing Architectures

The GARCH model was not used in isolation but was integrated into the broader framework of stock price prediction. Specifically:

- The **ARIMAX model** was used for predicting the mean returns.
- The **GARCH model** was applied to the residuals from ARIMAX to model volatility.
- The final output was a combination of the ARIMAX-predicted mean and the variance estimation from the GARCH model, providing a more comprehensive prediction of stock returns.

By integrating GARCH with ARIMAX, the model was able to separately handle both the trend and volatility components of stock returns, improving the overall robustness of the forecasting framework.

3.9 Stochastic Volatility Model with Gaussian Mixture Observation

3.9.1 Model Specification

The stochastic volatility (SV) model describes the evolution of the latent log-volatility x_t as an autoregressive process:

$$x_t = \phi_0 + \phi_1 x_{t-1} + \eta_t, \quad \eta_t \sim \mathcal{N}(0, \sigma_Q^2) \tag{3.1}$$

The observation equation is modeled as a two-component Gaussian mixture:

$$y_t \sim 0.5 \cdot \mathcal{N}(0, e^{x_t} + \sigma_{R0}^2) + 0.5 \cdot \mathcal{N}(\mu_1, e^{x_t} + \sigma_{R1}^2) \tag{3.2}$$

This mixture structure allows the model to capture heavy tails in financial return distributions, where $y_t = \log(r_t^2 + \varepsilon)$ represents the log-squared normalized return with a small constant ε to avoid numerical instability.

3.9.2 Filtering Procedure

We implement a recursive filtering algorithm to estimate the hidden state x_t at each time step:

1. **Prediction:**

$$x_t^- = \phi_0 + \phi_1 x_{t-1} \tag{3.3}$$

$$P_t^- = \phi_1^2 P_{t-1} + \sigma_Q^2 \tag{3.4}$$

2. **Compute Likelihoods:**

$$p_0 = \mathcal{N}(y_t; 0, e^{x_t^-} + \sigma_{R0}^2) \tag{3.5}$$

$$p_1 = \mathcal{N}(y_t; \mu_1, e^{x_t^-} + \sigma_{R1}^2) \tag{3.6}$$

3. **Posterior Weights:**

$$w_0 = \frac{0.5 \cdot p_0}{0.5 \cdot p_0 + 0.5 \cdot p_1}, \quad w_1 = 1 - w_0 \tag{3.7}$$

4. **Update:**

$$\text{residual} = w_0 \cdot \frac{y_t - 0}{\text{var}_0} + w_1 \cdot \frac{y_t - \mu_1}{\text{var}_1} \tag{3.8}$$

$$\text{gain} = \frac{P_t^-}{P_t^- + w_0 \cdot \text{var}_0 + w_1 \cdot \text{var}_1} \tag{3.9}$$

$$x_t = x_t^- + \text{gain} \cdot \text{residual} \tag{3.10}$$

This Kalman-like update allows efficient recursive estimation of the latent volatility under the non-Gaussian observation framework.

3.9.3 Parameter Estimation

Model parameters are estimated by minimizing the negative log-likelihood of the entire sequence using the L-BFGS-B optimization algorithm, subject to bounds ensuring stationarity and numerical stability.

3.10 Custom Ensemble Learning Architecture

3.10.1 Theory and Architecture Overview

This model is an ensemble learning framework designed to improve predictive accuracy by combining multiple base regressors and modeling their outputs using a neural network-based MetaLearner. The architecture consists of:

1. **Base Models:** Four tree-based regressors (XGBoost, LightGBM, CatBoost, RandomForest) and a time series model (ARIMAX).
2. **Meta Feature Generator:** Uses predictions from base models and their rolling MAE to construct meta features.
3. **MetaLearner:** A sequence-based LSTM model that learns to optimally combine meta features to predict the final target.

The ensemble leverages the strengths of both:

- *Tree-based models* for capturing nonlinear patterns.
- *ARIMAX* for linear autoregressive temporal relationships.
- *LSTM* for modeling sequential dependence in prediction errors and combining predictions dynamically.

3.10.2 Meta Feature Representation

Each input $\mathbf{x}_t \in \mathbb{R}^d$ is passed to the base models and ARIMAX to produce scalar predictions $\hat{y}_t^{(i)}$ for each model i. Along with each prediction, a rolling MAE score is computed:

$$\mathrm{MAE}_t^{(i)} = \frac{1}{w} \sum_{j=t-w+1}^{t} \left| y_j - \hat{y}_j^{(i)} \right| \tag{3.11}$$

The meta input $\mathbf{z}_t \in \mathbb{R}^{5\times 2}$ at time t is constructed as:

$$\mathbf{z}_t = \left[\left(\hat{y}_t^{(i)}, \mathrm{MAE}_t^{(i)} \right) \right]_{i=1}^{5} \tag{3.12}$$

where $i = \{\text{XGB, LGB, CAT, RF, ARIMAX}\}$.

3.10.3 MetaLearner Model

The MetaLearner is a neural network consisting of:

- **Input:** Sequence of shape $(5, 2)$
- **LSTM Layer:** 16 units with ReLU activation
- **Dense Output Layer:** Single scalar prediction

The output of the MetaLearner is given by:

$$\hat{y}_t^{\text{meta}} = \text{Dense}\left(\text{LSTM}(\mathbf{z}_t)\right) \tag{3.13}$$

3.10.4 Algorithm Flow

1. **Load and Split Dataset** into train, validation, and test sets.
2. **Train Base Models:**
    - XGBoost
    - LightGBM

- CatBoost
- RandomForest
- ARIMAX (on training set)

3. **Generate Meta Features:**
   - For each time step in train/val/test:
     - Predict using each base model
     - Forecast ARIMAX
     - Calculate rolling MAE of each model
     - Store $(\text{prediction}, \text{MAE})$ as meta feature
4. **Train MetaLearner (LSTM):**
   - Input: meta feature sequence $(5, 2)$
   - Output: predicted value $\hat{y}_t$
5. **Evaluate on Validation and Test Set** using RMSE

3.11 Transformer-based GAN Architecture

In this section, we propose a sequence-to-one forecasting framework inspired by the TTS-GAN architecture. The model consists of a Generator and a Discriminator, both utilizing Transformer Encoder blocks to process multivariate sequential data. The goal is to generate a future return value based on historical sequences, and adversarially train the model to ensure realistic output generation.

3.11.1 Generator Architecture

The Generator G takes as input a multivariate sequence $X \in \mathbb{R}^{T \times d}$ of length $T = 60$ with $d = 12$ features and predicts a scalar output $\hat{y}_{t+1} \in \mathbb{R}$. The input is first linearly projected into an embedding space and combined with learned positional encodings:

$$x'_t = \text{Dense}(X_t) + \text{Embedding}(t)$$

This representation is passed through L stacked Transformer Encoder blocks, followed by a Global Average Pooling operation and a final dense layer:

$$\hat{y}_{t+1} = G(X) = \text{Dense}(\text{GAP}(\text{TransformerBlocks}(x')))$$

3.11.2 Discriminator Architecture

The Discriminator D takes as input a univariate sequence of length 61: a real or generated scalar return $\hat{y}_{t+1}$ appended to the past 60 true values $y_{t-59}, \ldots, y_t$. The architecture is similar to the Generator: a dense projection followed by positional embeddings and multiple Transformer blocks. The final output is a scalar in $[0, 1]$:

$$D(Z) = \sigma(\text{Dense}(\text{GAP}(\text{TransformerBlocks}(Z))))$$

where $Z \in \mathbb{R}^{61 \times 1}$ is the sequence and σ denotes the sigmoid activation.

3.11.3 Transformer Encoder Block

Each Transformer Encoder block consists of a Multi-Head Self-Attention (MHSA) layer followed by a feed-forward network (FFN):

$$\text{MHSA}(X) = \text{LayerNorm}(X + \text{Dropout}(\text{MultiHeadAttention}(X, X)))$$

$$\text{FFN}(X) = \text{LayerNorm}(\text{MHSA}(X) + \text{Dropout}(\text{Dense}(\text{GELU}(\text{Dense}(X)))))$$

3.11.4 Training Procedure

The model is trained using a combination of mean squared error (MSE) and adversarial loss. Let y_{t+1} be the true target and $\hat{y}_{t+1}$ the generator's prediction. The Discriminator loss is given by:

$$\mathscr{L}_D = \text{BCE}(1, D(y_{\text{seq}}^{\text{real}})) + \text{BCE}(0, D(y_{\text{seq}}^{\text{fake}}))$$

where $y_{\text{seq}}^{\text{real}} = [y_{t-59}, \ldots, y_t, y_{t+1}]$ and $y_{\text{seq}}^{\text{fake}} = [y_{t-59}, \ldots, y_t, \hat{y}_{t+1}]$.

The Generator is optimized using:

$$\mathscr{L}_G = \text{MSE}(y_{t+1}, \hat{y}_{t+1}) + \lambda \cdot \text{BCE}(1, D(y_{\text{seq}}^{\text{fake}}))$$

where λ is a small weighting factor for adversarial loss (e.g., $\lambda = 0.01$).

3.11.5 Optimization

Gradients are computed using backpropagation through both the Generator and Discriminator. Separate optimizers are used for G and D, and their parameters are updated iteratively at each training step.

3.12 RAGIC Architecture for Financial Time Series Forecasting

3.12.1 Overview

The **RAGIC (Risk-Aware Generator with Integrated Critic)** architecture is a hybrid deep learning framework developed to model financial time series data with enhanced robustness and generalization. It integrates noise-injected attention mechanisms, temporal modeling using multi-head attention and TCNs, and adversarial learning using a Wasserstein critic to improve prediction performance on volatile data.

RAGIC is designed to forecast the next value in a sequence (e.g., stock return or price) by conditioning on multivariate historical data. It consists of three main modules:

- **Risk Module** (attention + noise for uncertainty modeling)
- **Temporal Module** (Transformer + TCN for time series understanding)
- **Critic** (Wasserstein GAN discriminator to enforce realistic prediction quality)

3.12.2 Theory and Components

Risk Module

The **Risk Module** aims to model market uncertainty by injecting Gaussian noise into the input sequence and applying attention-based weighting. This mechanism allows the model to focus on uncertain and potentially volatile patterns.

Given input $X \in \mathbb{R}^{W \times K}$ and noise $\varepsilon \sim \mathcal{N}(0, I)$ of shape $\mathbb{R}^{W \times 1}$, the steps are:

1. Concatenate: $X_{\text{aug}} = [X \mid \varepsilon]$
2. Attention weights: $a = \text{softmax}(WX_{\text{aug}})$
3. Weighted input: $\tilde{X} = a \odot X$
4. Residual connection and normalization: $X_{\text{risk}} = \text{LayerNorm}(\tilde{X} + X)$

Temporal Module

The **Temporal Module** processes the output of the risk module through a combination of positional embeddings, multi-head self-attention, and a two-layer Temporal Convolutional Network (TCN). It captures both long-term dependencies and local temporal dynamics.

Let $X_{\text{risk}} \in \mathbb{R}^{W \times K}$ be the input:

1. Add positional embedding: $X_{\text{pos}} = X_{\text{risk}} + \text{Embed}(\text{position})$
2. Multi-head attention: $X_{\text{attn}} = \text{MultiHeadAttention}(X_{\text{pos}})$
3. Residual + LayerNorm: $X_{\text{norm}} = \text{LayerNorm}(X_{\text{attn}} + X_{\text{pos}})$
4. TCN: Two dilated Conv1D layers with causal padding and ReLU activation.
5. Global pooling and dense output: $\hat{y} = \text{Dense}(\text{GlobalAvgPool}(X_{\text{tcn}}))$

Generator and Critic

Generator: Composed of the Risk and Temporal modules, it predicts the next time-step value $\hat{y}$ from a sequence X and random noise ε.

Critic: A GRU-based Wasserstein discriminator that receives a full sequence of W time steps (including either the predicted or real value as the last step) and outputs a scalar "realness" score:

$$D(X) = \text{Dense}(\text{GRU}(X))$$

3.12.3 Training Algorithm

The RAGIC framework is trained in an adversarial manner, where the Generator aims to produce realistic predictions, and the Critic learns to distinguish between real and generated sequences. The loss functions are:

- **Generator Loss (Predictor):**

$$\mathscr{L}_G = \text{MSE}(y, \hat{y})$$

- **Critic Loss:**

$$\mathscr{L}_D = \mathbb{E}_{\text{fake}}[D(X_{\text{fake}})] - \mathbb{E}_{\text{real}}[D(X_{\text{real}})]$$

3.12.4 Workflow

1. Load time series data $X \in \mathbb{R}^{T \times K}$ and target $Y \in \mathbb{R}^T$.
2. Create overlapping sequences of length $W = 60$.
3. Split into training and test sets.
4. For each epoch:
    - Generate noise ε for each batch.

- **Train Generator:**
  - Pass X and ε through Risk $\rightarrow$ Temporal to get $\hat{y}$.
  - Minimize MSE with ground truth y.
- **Train Critic:**
  - Concatenate predicted or real y with $X[:, 1:, :]$ to form fake/real sequences.
  - Maximize $\mathscr{L}_D$ (Wasserstein loss).

5. Track losses and visualize performance.

3.12.5 Applications

The RAGIC architecture is designed for time series forecasting tasks with inherent uncertainty and noise, such as:

- Stock return forecasting
- Volatility prediction
- Economic indicator forecasting

Its integration of noise-aware attention and adversarial learning enables better modeling of complex financial dynamics.

Chapter 4

Results and Evaluation

4.1 Evaluation Metrics and Criteria

Root Mean Squared Error (RMSE)

The Root Mean Squared Error (RMSE) is a criterion used to evaluate the accuracy of the model's forecast. It is calculated as the square root of the arithmetic mean of the squared differences between actual and predicted values. RMSE provides insights into the scale of errors, especially for regression models. It is defined as:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{4.1}$$

where:

- n: Number of observations
- y_i: Actual value of the i^{th} observation
- $\hat{y}_i$: Predicted value of the i^{th} observation

R-squared (R^2)

The coefficient of determination, R^2, measures the proportion of variance in the dependent variable explained by the model. Its range is from 0 to 1, where 1 indicates that the model explains all the variation, and 0 indicates no explanatory power. It is defined as:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{4.2}$$

where:

- n: Number of observations
- y_i: True value of the i^{th} observation
- $\hat{y}_i$: Predicted value of the i^{th} observation
- $\bar{y}$: Mean of the actual values

4.2 Performance of Enhanced Hybrid LSTM-RLS Architecture

4.2.1 Results Before and After Shift Adjustment

Table 4.1: LSTM-RLS Performance Metrics

LSTM-RLS	Daily Returns	Weekly Returns	Daily Close
RMSE	0.76%	1.44%	161.8935
R2	-0.0636	-0.038	0.9956

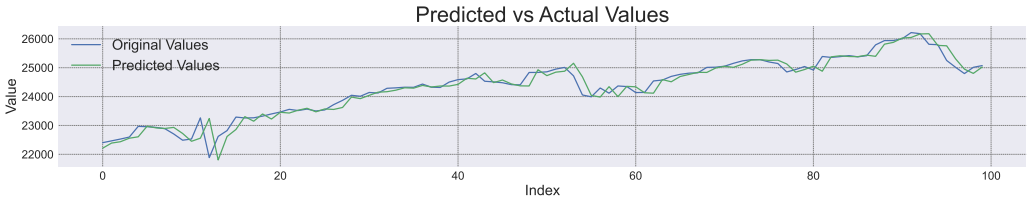

Figure 4.1: Original vs. Predicted Stock Prices

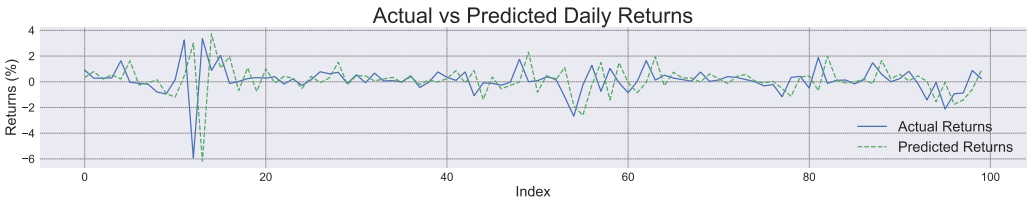

Figure 4.2: Returns: Original vs. Predicted Prices

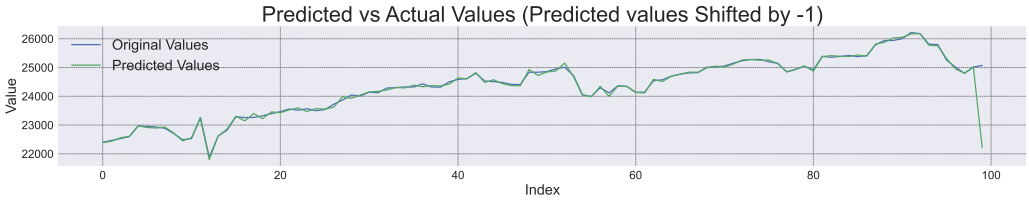

Figure 4.3: Stock Prices: Shifted Predictions (-1)

Table 4.2: RMSE Comparison for Daily Returns (LSTM-RLS)

LSTM-RLS Error between Actual and Predicted Daily Returns	
Original RMSE	1.01%
RMSE with Shift - 1	**0.24%**

4.2.2 Performance with Multi-Feature Dataset

Table 4.3: Performance Metrics for Multi-Feature LSTM-RLS

Metric	**Value**
RMSE	0.91%
R^2 Score	-0.061

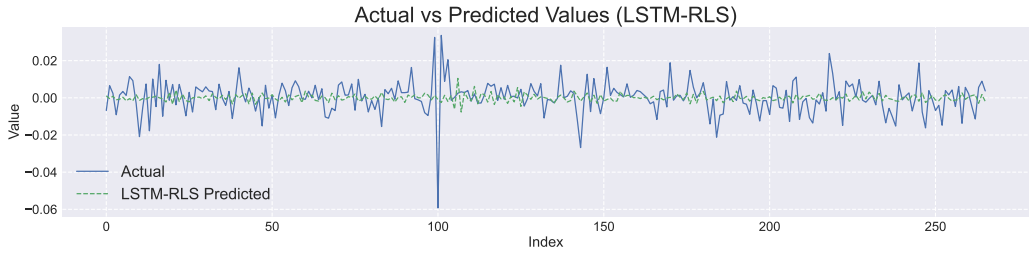

Figure 4.4: Stock Prices: Actual vs. Predicted Daily Returns

4.3 Performance of DNS Architecture

Figure 4.5 shows the best actual vs predicted plot when trained and tested on the DNS architecture to remove lag in prediction.

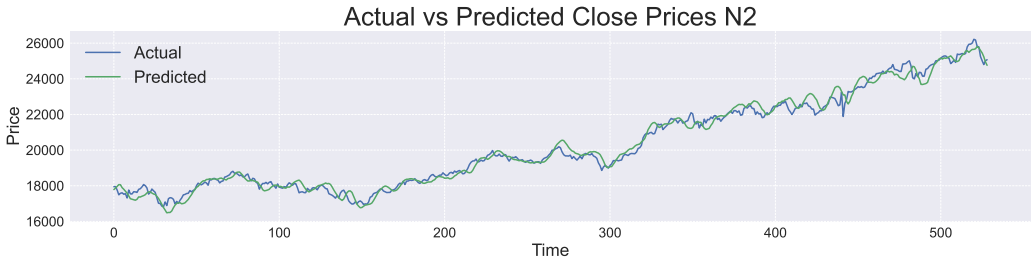

Figure 4.5: Comparison of Stock Prices and Returns (Shifted -1)

Table 4.4 shows the RMSE and R2 results for the DNS architecture with all combination of networks used.

Table 4.4: Comparison of DNS Models: LSTM-SDM, LSTM-MA, RBF-SDM, and RBF-MA

DNS	**LSTM-SDM**	**LSTM-MA**	**RBF-SDM**	**RBF-MA**
RMSE	1082.0187	688.529	**328.7387**	479.8617
R2	0.8092	0.9227	**0.9823**	0.9624

4.4 ARIMA-LSTM Residual Framework Results

4.4.1 ARIMAX Test Predictions

Figures 4.6 and 4.7 present the test predictions from the ARIMAX model when the output variable is close price and daily returns, respectively.

Figure 4.6: ARIMAX Test Predictions - Output: Close Price

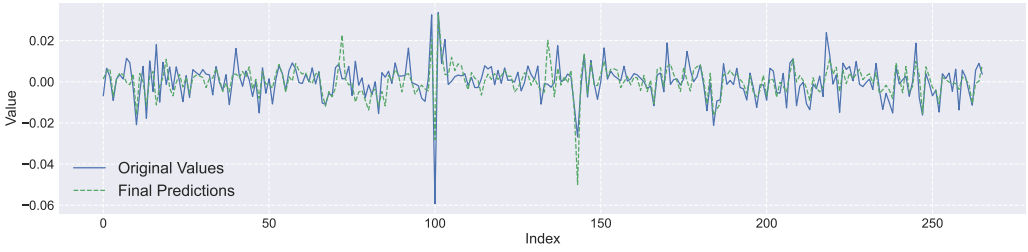

Figure 4.7: ARIMAX Test Predictions - Output: Returns

4.4.2 Final Test Predictions

Figures 4.8 and 4.9 illustrate the final test predictions after integrating LSTM when the output is close price and daily returns.

4.4.3 Performance Metrics

Table 4.5 compares the test RMSE and R^2 scores for ARIMAX and the final model when predicting close prices and daily returns.

Figure 4.8: Final Test Predictions - Output: Close Price

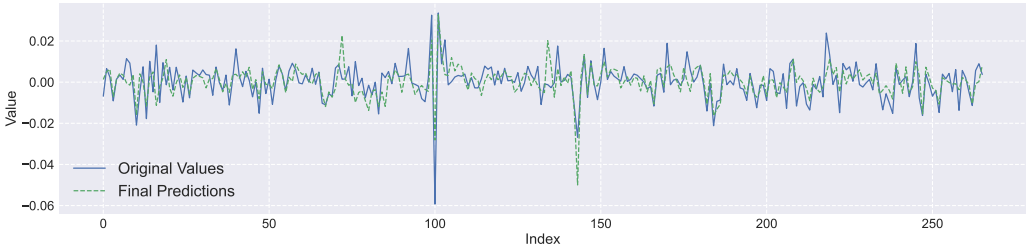

Figure 4.9: Final Test Predictions - Output: Returns

Table 4.5: Test RMSE and R^2 Score for ARIMAX and Final Model

Model	**RMSE**	**R^2 Score**
ARIMAX (Close)	7431.43	-35.47
ARIMAX (Returns)	0.703%	0.36
Final Model (Close)	7394.61	-35.11
Final Model (Returns)	0.669%	0.42

4.4.4 Impact of RLS on Final Predictions

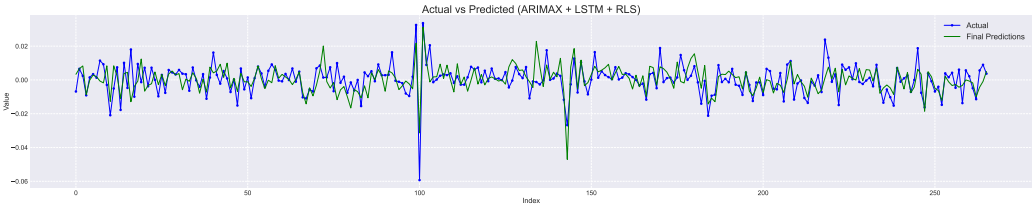

Figure 4.10: Actual vs. Predicted Returns After Applying RLS to Final Predictions

Table 4.6: Performance Metrics After Applying RLS to Residuals

Metric	Value
RMSE	0.72%
R^2 Score	0.32

4.5 Multi-Feature LSTM Results

This section presents the evaluation results of the Multi-Feature LSTM Forecasting Framework, focusing on its performance in predicting daily close returns. The evaluation metrics and visualizations demonstrate the accuracy and adaptability of the framework, as well as its ability to handle real-time updates as per the incremental training methodology discussed earlier.

4.5.1 Evaluation Metrics and Analysis

The framework's performance is evaluated using two key visualizations:

1. **Absolute Error vs Total RMSE:** Figure 4.11 plots the absolute error between actual and predicted daily close returns in one time step prediction. Besides, it plots the total RMSE of the test set for the entire duration of evaluation. This graph plots how well the architecture minimizes the errors from predictions.

2. **Actual vs Predicted Close Returns:** Figure 4.12 depicts the agreement Between actual and predicted daily close returns with re-training of the model after every prediction following incremental training approach. The above plot indicates the learning ability of the model as well as real-time predictive accuracy.

4.5.2 Interpretation of Results

- The high absolute error values observed in Figure 4.11 verify that the model is not very good at predicting daily close returns.
- The difference between the actual and predicted returns in Figure 4.12 is a result of how the incremental training works, allowing the model to capture recent market trends.
- A comparison of absolute error with total RMSE clearly highlights the significance of one-time step prediction error minimization for successful application in algorithmic trading.

4.5.3 Visualization of Results

The plots illustrating the framework's performance are provided below:

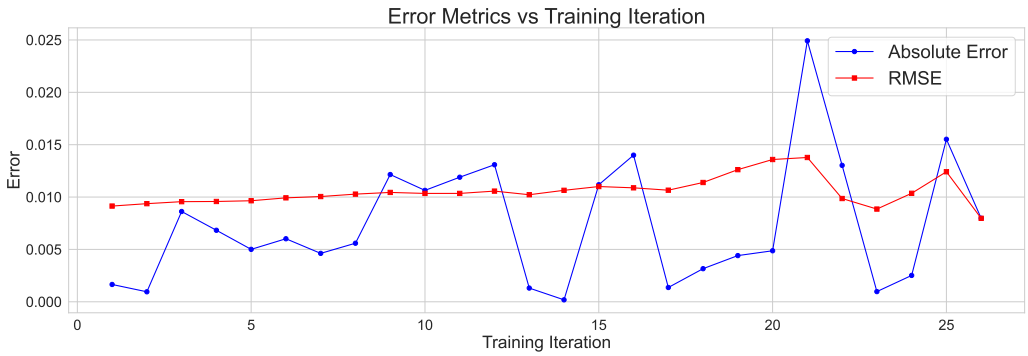

Figure 4.11: Absolute error between actual and predicted one-time step daily close return, compared with the total RMSE of the test data.

4.6 GARCH Model Results

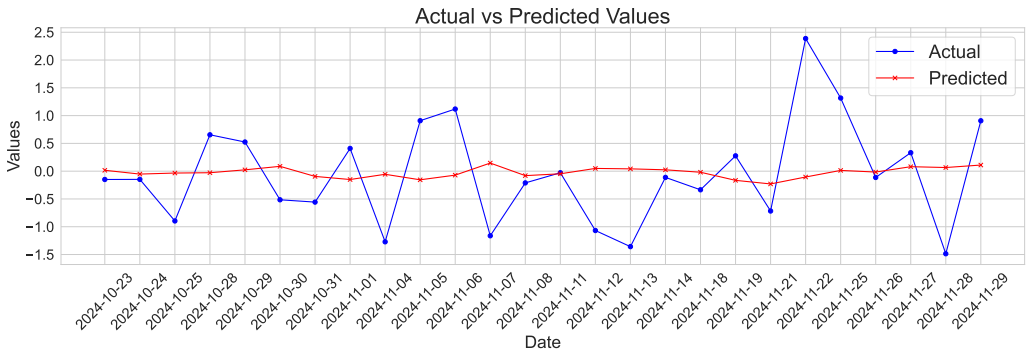

Figure 4.12: Actual vs Predicted daily close return where the model is retrained after each prediction as per the incremental training methodology.

4.6.1 Evaluation Metrics and Performance

The GARCH model does not capture any volatility from the residuals, resulting in final predictions that are nearly identical to the ARIMAX model outputs.

4.6.2 Residuals from ARIMAX Predictions

To analyze the deviations captured by the GARCH model, we plot the residuals obtained by subtracting ARIMAX predictions from the actual values.

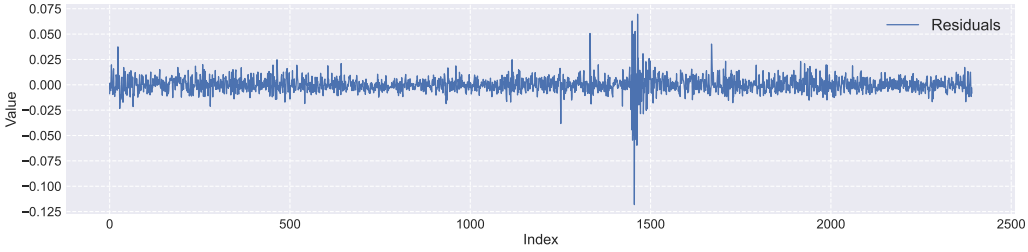

Figure 4.13: Residuals: Actual Values - ARIMAX Predictions

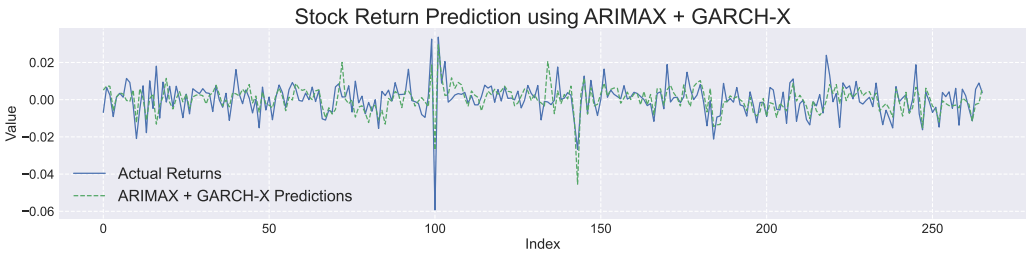

Figure 4.14: Actual vs. Predicted Returns Using GARCH Model

Table 4.7: Performance Metrics of GARCH Model

Metric	**Value**
RMSE	0.703%
R^2 Score	0.36

4.6.3 Visualization of Predictions

4.7 Stochastic Volatility Results

4.7.1 Actual vs Predicted Returns

Figure 4.15 shows the normalized actual returns plotted against the normalized filtered log-volatility estimates from the stochastic volatility model.

4.7.2 Distribution Comparison

Figure 4.16 displays the kernel density estimation (KDE) curves of actual returns versus fitted returns from the stochastic volatility model, highlighting how well the model captures the return distribution characteristics.

4.7.3 Performance Metrics

- **RMSE:** 0.0104

- **R^2 Score:** -0.2091

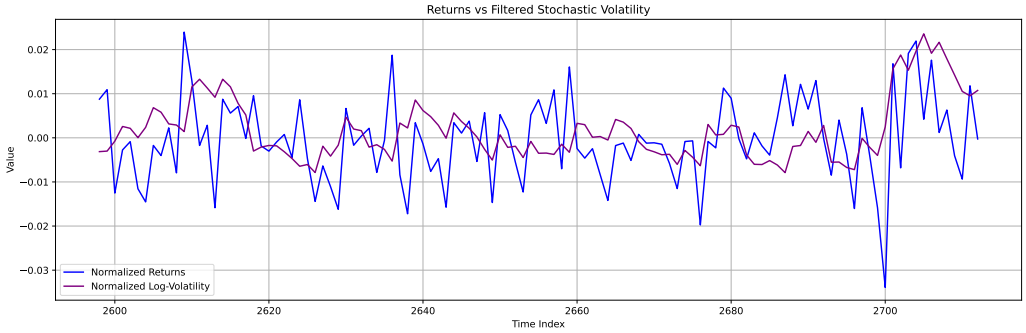

Figure 4.15: Normalized actual returns vs filtered log-volatility (Stochastic Volatility Model).

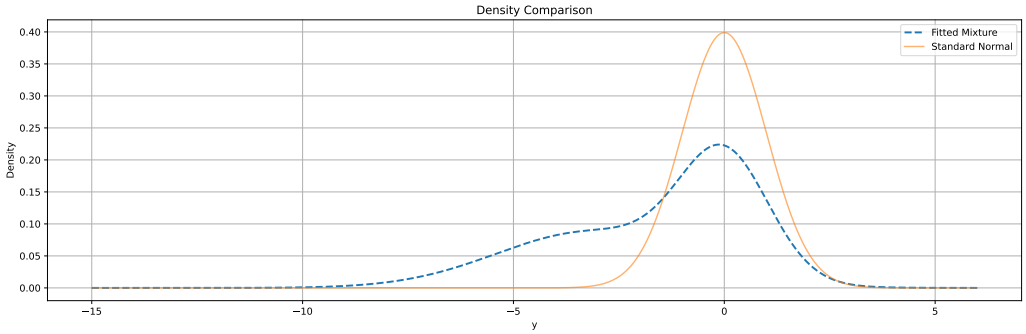

Figure 4.16: Density comparison of actual returns vs fitted returns (Stochastic Volatility Model).

4.8 Ensemble Learning Results

4.8.1 Actual vs Predicted Returns

Figure 4.17 presents the predicted daily returns from the ensemble meta-learner compared with the actual close returns.

4.8.2 Performance Metrics

- **MetaNet RMSE (Validation):** 0.0065
- **MetaNet RMSE (Test):** 0.0064
- **MetaNet R^2 Score (Test):** —-

4.8.3 Base Model Comparison

Table 4.8 shows the RMSE values for each base model on both the validation and test sets, highlighting the performance of MetaNet as the final predictor.

Table 4.8: RMSE of Base Models and MetaNet (Validation and Test Sets)

Model	RMSE (Validation)	RMSE (Test)
XGBoost	0.00670	0.00624
LightGBM	0.00670	0.00638
CatBoost	0.00663	0.00636
RandomForest	0.00667	0.00658
ARIMAX	0.00713	0.00766
MetaNet	**0.00655**	**0.00638**

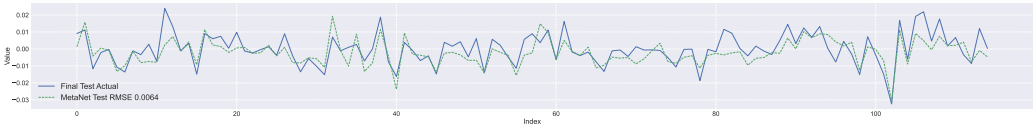

Figure 4.17: Actual vs Predicted daily close return (Ensemble Meta-Learner).

4.9 Transformer GAN Results

4.9.1 Actual vs Predicted Returns

Figure 4.18 illustrates the ability of the Transformer-based GAN to replicate daily return dynamics.

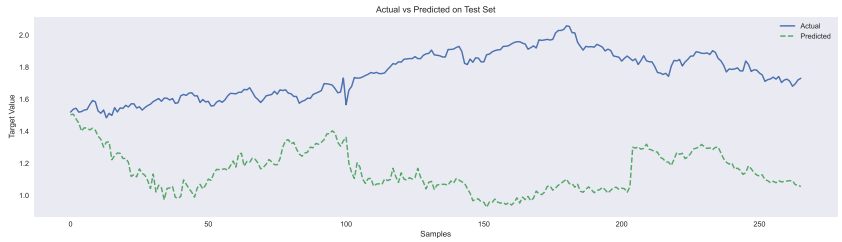

Figure 4.18: Actual vs Predicted daily close return (Transformer GAN).

4.10 RAGIC Framework Results

4.10.1 Actual vs Predicted Returns

Figure 4.19 compares the predicted returns from the RAGIC framework against the actual returns, reflecting the model's interpretability and precision.

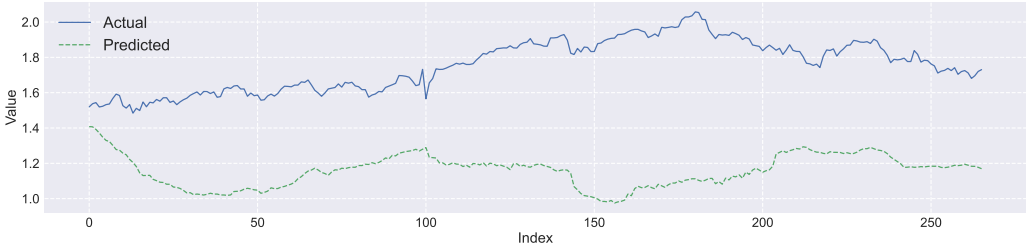

Figure 4.19: Actual vs Predicted daily close return (RAGIC Framework).

Chapter 5

Discussions and Challenges

5.1 Key Findings

Although the RMSE and R^2 score of the enhanced hybrid LSTM-RLS model were lower, the actual vs. prediction plot revealed a constant lag of one time step in the predicted stock prices. To address this issue, the DNS architecture was adopted. However, despite experimenting with various combinations of activation functions and slope trend methods, the model could not eliminate the lag.

Consequently, the DNS architecture was replaced by the ARIMA-LSTM Residual Integration Framework. Since ARIMA did not perform as expected in predicting the linear component, LSTM was utilized to predict both linear and non-linear components. However, this model also failed to remove the lag and exhibited a higher RMSE compared to other model architectures.

To further enhance performance, additional features were incorporated, and a dataset with multiple features was developed as described earlier. The Multi-Feature LSTM Forecasting Framework was then employed. Despite these en-

hancements, this model architecture, when combined with the proposed training method, was also unable to eliminate the lag in predictions.

Additionally, the Hybrid LSTM-RLS with a multi-feature input did not perform well. The reason behind this is that the LSTM output, which consists of daily returns, is highly volatile in nature, making it difficult for the model to generalize effectively.

The ARIMAX-LSTM-RLS model also slightly reduced performance, suggesting that the final predictions obtained from the residuals architecture do not serve as suitable inputs for the RLS model in an online prediction setting.

Furthermore, GARCH, for some reason, failed to capture any volatility (variance) in the residuals obtained after the mean predictions by ARIMAX. As a result, the final predictions produced by GARCH were identical to those obtained from ARIMAX alone, rendering GARCH ineffective in improving prediction performance.

Based on the comparative performance of different architectures tested in this dissertation, the residuals architecture has demonstrated the best predictive capability so far.

In the case of the Stochastic Volatility model, although lags were observed at certain points, the filtered volatility estimates closely followed the overall return patterns with moderate error, demonstrating the model's ability to track dynamic volatility behavior.

For the Ensemble Learning Framework, predictions from ARIMAX, Random Forest, CatBoost, LightGBM, and XGBoost were combined using an LSTM-based MetaNet. This approach successfully eliminated the one-time step lag and closely followed the actual return patterns. However, occasional directional

mispredictions led to slightly elevated RMSE values. Despite this, the ensemble method outperformed all other architectures tested in this study.

The Transformer GAN-based architecture and the RAGIC model, implemented directly as described in their respective papers, showed significantly higher prediction errors. Even after extended training (up to 500 epochs), both architectures struggled to generalize. Due to limited time, these GAN-based models were not significantly modified, which might have contributed to their underperformance.

To conclude, among all the tested models and algorithms, the Ensemble Learning Framework delivered the best overall performance in terms of pattern tracking, lag elimination, and balanced error metrics.

5.2 Challenges and Model Refinement

A significant challenge faced during the development of the hybrid models was the persistent lag in long-term predictions. Despite repeated efforts to refine the models—particularly the hybrid LSTM-RLS architectures—the lag in predictions continued to appear consistently. While the introduction of alternative models aimed at mitigating this lag did result in lower RMSE values in some cases, the core issue of delayed predictions remained unresolved.

This underscores the difficulty in simultaneously achieving high accuracy and eliminating temporal lag, especially in sequential financial data. The lag persisted even with models explicitly designed for short-term responsiveness, highlighting a fundamental trade-off between complexity and real-time adaptability.

Volatility modeling through GARCH also posed substantial challenges. Despite being a well-established method, GARCH failed to capture any meaningful

variance in the ARIMAX residuals. The model consistently predicted near-zero volatility, rendering it ineffective in enhancing the prediction accuracy. This unexpected outcome underlines the limitations of applying classical volatility models to highly noisy and non-linear residual structures in financial time series.

The Stochastic Volatility framework, although able to capture the broad shape and dynamics of volatility, occasionally exhibited lag and divergence from actual return magnitudes. This suggests that while it is effective in estimating underlying variance trends, aligning those estimates with return dynamics remains a complex task.

Ensemble Learning, though the most successful overall, brought its own set of challenges. Integrating predictions from multiple diverse models—including ARIMAX, Random Forest, CatBoost, LightGBM, and XGBoost—into a MetaNet required careful feature normalization, input alignment, and consistent retraining to ensure stability. Additionally, occasional directional mispredictions contributed to local spikes in error despite the overall reduction in lag.

Implementation of Transformer GAN and RAGIC architectures also presented technical and methodological difficulties. Due to time constraints, these models were implemented with minimal modifications to their original architectures. Despite extensive training, they showed high prediction errors and poor generalization. The complexity of GAN training and sensitivity to hyperparameter tuning further limited their effectiveness in this study.

In summary, while significant advancements were made in predictive modeling of stock returns, persistent challenges such as time-step lag, ineffective volatility modeling, and difficulties in training generative architectures remain unresolved. These limitations highlight areas for future research, particularly in

adaptive modeling, robust volatility estimation, and customized GAN architectures for financial time series forecasting.

Chapter 6

Conclusion and Future Direction

6.1 Conclusion

This work explored multiple hybrid and deep learning-based models to improve stock return prediction. The initial Hybrid LSTM-RLS model showed consistent one-step lag, which persisted across the DNS and ARIMA-LSTM Residual Integration frameworks, despite minor RMSE improvements. The Multi-Feature LSTM Framework improved input representation but failed to resolve the lag issue.

GARCH integration over ARIMAX residuals proved ineffective, consistently predicting near-zero volatility. The Stochastic Volatility model followed general return patterns but exhibited lags and deviations. Transformer GAN and RAGIC architectures, implemented with minimal tuning, resulted in high prediction errors.

The Ensemble Learning Framework, combining ARIMAX, tree-based models, and a MetaNet, delivered the best performance by eliminating lag and closely

tracking return trends, despite some directional prediction errors. Overall, ensemble learning proved the most effective strategy for real-time stock return forecasting among all models tested.

6.2 Future Direction

Future work will focus on exploring alternative architectures and advanced techniques to address the issue of lag and further improve the performance of stock price prediction models. Additionally, incorporating more sophisticated feature engineering and model tuning strategies could enhance prediction accuracy and help overcome the challenges faced in this study.

Chapter 7

Publications

The paper based on this work was presented at the **28th Nirma International Conference on Management (NICOM)**, held from **January 8 to January 10, 2025**. The presentation was part of the sub-theme **Finance and Accounting**, under the track titled "*Financial Technologies and Digitalization: Financial Analytics and Machine Learning & Deep Learning Applications in Finance.*"

Based on the research conducted in this dissertation, the following journals have been identified as suitable targets for future publication:

1. The Journal of Finance and Data Science

- **Impact Factor:** 3.9
- **Scopus Indexed:** Yes

2. Data Science in Finance and Economics

- **Impact Factor:** 1.3
- **Scopus Indexed:** No

These journals have been selected based on their relevance to the subject areas of finance, time series forecasting, and data science.

Bibliography

[1] A. K. Dastgerdi and P. Mercorelli, "Investigating the effect of noise elimination on LSTM models for financial markets prediction using kalman filter and wavelet transform," *WSEAS TRANSACTIONS ON BUSINESS AND ECONOMICS*, vol. 19, pp. 432–441, Jan. 18, 2022, ISSN: 2224-2899, 1109-9526. DOI: 10.37394/23207.2022.19.39. [Online]. Available: https://wseas.com/journals/bae/2022/a765107-015(2022).pdf (visited on 07/10/2024).

[2] W. Fang, W. Cai, B. Fan, J. Yan, and T. Zhou, "Kalman-LSTM model for short-term traffic flow forecasting," in *2021 IEEE 5th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, Chongqing, China: IEEE, Mar. 12, 2021, pp. 1604–1608, ISBN: 978-1-72818-028-1. DOI: 10.1109/IAEAC50856.2021.9390991. [Online]. Available: https://ieeexplore.ieee.org/document/9390991/ (visited on 06/11/2024).

[3] F. Song, Y. Li, W. Cheng, L. Dong, M. Li, and J. Li, "An improved kalman filter based on long short-memory recurrent neural network for nonlinear radar target tracking," *Wireless Communications and Mobile Computing*, vol. 2022, A. Alamoodi, Ed., pp. 1–10, Aug. 21, 2022, ISSN: 1530-8677,

1530-8669. DOI: 10.1155/2022/8280428. [Online]. Available: https://www.hindawi.com/journals/wcmc/2022/8280428/ (visited on 06/11/2024).

[4] Y. Tian, Z. Lian, P. Wang, M. Wang, Z. Yue, and H. Chai, "Application of a long short-term memory neural network algorithm fused with kalman filter in UWB indoor positioning," *Scientific Reports*, vol. 14, no. 1, p. 1925, Jan. 22, 2024, ISSN: 2045-2322. DOI: 10.1038/s41598-024-52464-y. [Online]. Available: https://www.nature.com/articles/s41598-024-52464-y (visited on 06/11/2024).

[5] C. Wang, R. Li, Y. Cao, and M. Li, *A hybrid model for state of charge estimation of lithium-ion batteries utilizing improved adaptive extended kalman filter and long short-term memory neural network*, 2024. DOI: 10.2139/ssrn.4834979. [Online]. Available: https://www.ssrn.com/abstract=4834979 (visited on 07/05/2024).

[6] H. Shah, V. Bhatt, and J. Shah, "A neoteric technique using ARIMA-LSTM for time series analysis on stock market forecasting," in *Mathematical Modeling, Computational Intelligence Techniques and Renewable Energy*, M. Sahni, J. M. Merigó, R. Sahni, and R. Verma, Eds., vol. 1405, Series Title: Advances in Intelligent Systems and Computing, Singapore: Springer Singapore, 2022, pp. 381–392, ISBN: 9789811659515 9789811659522. DOI: 10.1007/978-981-16-5952-2_33. [Online]. Available: https://link.springer.com/10.1007/978-981-16-5952-2_33 (visited on 07/31/2024).

[7] J. Yoon and D. Jarrett, "Time-series generative adversarial networks,"

[8] J. Gu, W. Du, and G. Wang, "RAGIC: Risk-aware generative framework for stock interval construction," *IEEE Transactions on Knowledge and Data Engineering*, vol. 37, no. 4, pp. 2085–2096, Apr. 2025, ISSN: 1041-4347, 1558-2191, 2326-3865. DOI: 10.1109/TKDE.2025.3533492. [Online]. Available: https://ieeexplore.ieee.org/document/10885039/ (visited on 04/11/2025).

[9] S. Samanta, M. Pratama, S. Sundaram, and N. Srikanth, "A dual network solution (DNS) for lag-free time series forecasting," in *2020 International Joint Conference on Neural Networks (IJCNN)*, Glasgow, United Kingdom: IEEE, Jul. 2020, pp. 1–8, ISBN: 978-1-72816-926-2. DOI: 10.1109/IJCNN48605.2020.9207022. [Online]. Available: https://ieeexplore.ieee.org/document/9207022/ (visited on 08/05/2024).