Artificial Intelligence in Stock Market Prediction: A Deep Learning Perspective

Nikhil Adapala & Aleena Shabbir

Received February 17, 2025
Accepted May 24, 2025
Electronic access June 30, 2025

Accurately forecasting stock prices remains a challenging task due to the volatile nature of financial markets and the vast volume of influencing factors. This study investigates the effectiveness of various modeling approaches in predicting stock prices, aiming to improve upon traditional methods by integrating deep learning techniques and alternative data sources. The primary objective is to compare the predictive power of traditional statistical models with modern neural network architectures using both numerical and sentiment-based inputs. A comprehensive dataset combining historical stock data, macroeconomic indicators, Google Trends, and Twitter sentiment was utilized. Models tested include ARIMA, Bidirectional LSTM (BiLSTM), Convolutional Neural Network (CNN), Transformer, a BERT-inspired architecture for numerical input, and a custom Hybrid model. Each model was trained using a five-day sliding window approach and optimized via Keras Tuner. Performance was evaluated using MAE, MSE, MAPE, and R metrics. Results show that BiLSTM and CNN models significantly outperformed ARIMA, particularly when sentiment data was incorporated. While Transformer and BERT-based models demonstrated strong results, the improvement over BiLSTM and CNN was marginal. The Hybrid model underperformed despite its complexity, but ablation studies revealed key insights into component contributions. This study highlights the importance of model selection, data diversity, and architecture simplicity in financial forecasting, offering practical guidance for future research and real-world applications.

Keywords: Stock market forecasting, Deep learning, Statistical models, Transformers, Sentiment analysis, Google Trends.

1 Introduction

The forecasting of stock market prices is, and has been, a significant challenge in both finance and machine learning. The stock market is central to the global economy, serving as a potential source of income for individuals and institutions. However, stock market forecasting remains difficult due to its inherent volatility. Factors such as public perception, economic indicators, social media sentiment, and political events can influence prices[1]. Without highly powerful machines, it is nearly impossible for the average individual to keep up with large institutions that benefit from insider information. Enter stock market forecasting: the idea of analyzing past market movements and influencing factors to forecast future prices. In the past, forecasting depended on traditional statistical models such as Auto-Regressive Integrated Moving Average (ARIMA) and Generalized Auto-Regressive Conditional Heteroskedasticity (GARCH), offering interpretability, efficiency, and the ability to model short-term volatility[2]. Despite their strengths, these models assumed linearity, limiting their effectiveness in complex, nonlinear systems[3]. To address these limitations, researchers have explored deep learning architectures. Models such as Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks are adept at capturing temporal dependencies, while Convolutional Neural Networks (CNNs) are effective at detecting short-term patterns[4]. More recently, attention-based models such as Transformers have gained popularity for their ability to process long sequences efficiently[5]. However, they remain sensitive to overfitting and require large volumes of data[6]. To overcome these challenges, this study introduces two novel architectures. The first is a Hybrid model that integrates the strengths of Bidirectional LSTMs, CNNs, and Transformers to capture both short-term and long-term patterns in stock data[7]. The second is a BERT-inspired model, adapted from the natural language processing domain, designed to handle numerical time-series data by learning contextual relationships across time steps[8]. The models were trained a comprehensive dataset that includes not only stock indicators but also sentiment scores (generated using the FinBERT-tone model[9], macroeconomic indicators from the World Bank DataBank[10], and Google Trends data[11], covering the period from January 2015 to September 2019. Data for companies such as Tesla and Meta were included where applicable. Models were evaluated using a sliding window approach in which each model was provided five consecutive days of market data and tasked with forecasting the sixth days closing price[1]. Although multiple window sizes between 3 and 50 days were tested, a 5-day window was selected as the optimal configuration based on validation performance[3].

This paper compares the performance of ARIMA, Bidirectional LSTM, CNN, Transformer, Hybrid, and BERT-inspired models using standard regression metrics including Mean Absolute Error (MAE), Mean Squared Error (MSE), Mean Absolute Percentage Error (MAPE), and the Coefficient of Determination (R^2)[1,12,13]. By testing a range of architectures, this study provides a comprehensive analysis of the current capabilities of deep learning in financial forecasting and outlines promising directions for future research.

2 Literature Review

Stock market prediction is a longstanding area of interest in both financial and computational research due to the inherently volatile, nonlinear, and multifactorial nature of market behavior. Early approaches relied on statistical models such as Auto-Regressive Integrated Moving Average (ARIMA) and Generalized Auto-Regressive Conditional Heteroskedasticity (GARCH), which offered interpretability and ease of implementation. However, these models assume linearity and stationarity, limiting their effectiveness in capturing the dynamic and complex structure of modern financial data[2,3]. To address these limitations, more recent studies have explored deep learning methods, which can model nonlinear relationships and temporal dependencies in large datasets. Long Short-Term Memory (LSTM) networks and their bidirectional variant (BiLSTM) have shown particular promise in financial forecasting. Their ability to retain information across time steps makes them well-suited for detecting trends in sequential stock data. Prior work by Siami-Namini et al. (2019) and Fischer and Krauss (2018) demonstrated the superior performance of LSTM-based models over traditional statistical approaches when applied to stock price prediction[12,13]. Convolutional Neural Networks (CNNs), originally developed for image processing, have also been successfully adapted for time series forecasting. CNNs are particularly effective in extracting short-term local patterns from financial data and have been shown to improve prediction performance when used alone or in conjunction with LSTM layers[4,14] . Building on the success of these models, Transformer-based architectures have emerged as a powerful alternative due to their attention mechanism, which enables them to model long-range dependencies without the need for sequential processing. Studies such as Zerveas et al. (2020) and Farsani and Pazouki (2021) have shown that Transformers outperform traditional recurrent models on a variety of time series tasks, including financial forecasting[6,15]. Additionally, there has been growing interest in combining models to create hybrid architectures that leverage the strengths of different components. Models that combine CNNs, LSTMs, and attention mechanisms have been explored in recent literature, showing potential to capture both short-term fluctuations and long-term dependencies[16,17]. However, these models often require careful tuning to avoid redundancy or instability in learning. Finally, the use of pre-trained language models such as BERT has expanded into numerical domains. Though originally designed for natural language processing, BERT's ability to learn contextual relationships has inspired adaptations for time series forecasting. These adaptations aim to apply self-attention to numerical sequences, potentially enhancing interpretability and performance in financial prediction tasks[8]. Given this body of work, the present study was designed to evaluate and compare a comprehensive set of forecasting modelsincluding ARIMA, BiLSTM, CNN, Transformer, Hybrid, and a BERT-inspired architectureusing a unified dataset that includes historical stock data, sentiment scores, macroeconomic indicators, and Google Trends. The selection of models reflects both their relevance in existing literature and their architectural diversity, allowing for a systematic evaluation of which approaches are most suitable for real-world stock market forecasting.

3 Dataset

To support this study, we constructed a comprehensive dataset by integrating data from four publicly available sources: stock market data from a MAAMA dataset, Twitter sentiment data, macroeconomic indicators from the World Bank DataBank, and Google Trends data. The final dataset spans from **January 2015 to December 2019** and includes data for four major technology companiesApple, Amazon, Google, and Microsoft. Each row in the dataset represents a unique company-date pair and is labeled with both the corresponding ticker symbol and company name. Stock market data, including the daily *open*, *high*, *low*, *close*, *adj_close*, and *volume* values, were directly sourced from the MAAMA dataset[18]. These fields reflect standard market performance indicators and are included without modification. The *next_open* value, representing the stocks opening price on the following trading day, was created as the prediction target for our models. To measure public sentiment, we used a Twitter dataset in which each tweet was labeled by its corresponding company and posting date[19]. We employed a pre-trained FinBERT-tone modelfine-tuned for financial sentiment analysisto evaluate each tweet and assign a sentiment score ranging from 1 (highly negative) to 1 (highly positive)[9]. We then aggregated these tweet-level scores into a single *weekly_sentiment_score* for each company by averaging the sentiment scores of all tweets associated with that company within each week. Only sentiment scores within the 20152019 date range were included in the final merged dataset. Macroeconomic context was provided by the World Bank DataBank[10]. We included six key indicators: *unemployment_rate*, *gdp*, *gdp_growth*, *gdp_per_capita*, *gdp_per_capita_growth*, and *inflation_rate*. These values were aligned with each row by matching annual data to the corresponding calendar year. To capture public interest over time, we used Google Trends data[11]. For each company, we collected monthly search interest scoresnormalized from 0 to 100using

the Google Trends platform. These values were averaged for each month from January 2015 to December 2019 and labeled as the *trend* score for the respective company. This score serves as a proxy for public attention and online interest. The final dataset combines market performance data, sentiment analysis, macroeconomic indicators, and online interest into a unified structure suitable for time-series forecasting and machine learning tasks.

List of Dataset Features

- **date**: The calendar date for the record.
- **open**: The stocks opening price on the given day.
- **high**: The highest price reached during the trading day.
- **low**: The lowest price reached during the trading day.
- **close**: The final price at market close.
- **adj_close**: The adjusted close accounting for splits and dividends.
- **volume**: Number of shares traded on that day.
- **unemployment_rate**: Annual unemployment rate as a percentage [10].
- **gdp**: Gross domestic product in current U.S. dollars [10].
- **gdp_growth**: Annual GDP growth rate [10].
- **gdp_per_capita**: GDP per person in current U.S. dollars [10].
- **gdp_per_capita_growth**: Annual GDP per capita growth rate [10].
- **inflation_rate**: Annual percentage change in consumer prices [10].
- **trend**: Monthly Google Trends score for public search interest [11].
- **weekly_sentiment_score**: Weekly sentiment score derived from tweets [9,19].
- **ticker**: Stock ticker symbol for the company.
- **next_open**: Opening stock price on the next trading day (target variable).

4 Methodology and Experiments

This section outlines the technical foundations, architectural design, and implementation strategies used in this study. We begin by introducing each forecasting model independently, ranging from traditional statistical techniques to advanced deep learning architectures. For each model, we describe its theoretical underpinnings, core mechanisms, and adaptations for time-series forecasting. We also detail how each model was trained and optimized, including the use of hyperparameter tuning. By organizing the section this way, we aim to provide a clear and replicable understanding of how each method was applied to the financial prediction task. To evaluate model performance, the dataset was split using an 80/20 train-test split, with 80% of the data used for training and 20% reserved as an unseen test set. Within the training set, Keras Tuner was employed for hyperparameter optimization using cross-validation on the training portion alone. The test set was kept entirely separate during this process and was not exposed to the model until final evaluation. This ensured that performance metrics reflect the model's ability to generalize to truly unseen data and that no information leakage occurred during tuning.

A. Auto-Regressive Integrated Moving Average (ARIMA) ARIMA is and has been for a long time, a traditional time-series forecasting model [2]. While designed for linear patterns, it has been widely used in financial analysis because of its simplicity, interpretability, and effectiveness in capturing short-term trends and sequences. This architecture joins three components:

- **Auto-Regression (AR)**: A regression model that analyzes the dependent relationship between the independent and a certain number of lagged dependent variables.
- **Integrated (I)**: Differencing the time-series data to make it stationary by removing trends or seasonality.
- **Moving Average (MA)**: Models the relationship between an observation and a residual error from a moving average model applied to lagged observations. These components are parameterized by (p, d, q), where:
  - **p**: Order of the AR term (number of lag observations used).
  - **d**: Order of differencing to achieve stationarity.
  - **q**: Order of the MA term (size of the moving average window).

The Autoregressive Integrated Moving Average (ARIMA) model is a classical statistical technique for univariate time series forecasting. It combines three components: autoregression (AR), differencing (I), and moving average (MA), defined

respectively by the parameters p, d, and q. The general form of the ARIMA(p,d,q) model is given as:

$$\Phi(L)(1-L)^d y_t = \Theta(L)\varepsilon_t \tag{1}$$

In Equation 1:

- L is the lag operator, where $L^k y_t = y_{t-k}$
- $(1-L)^d$ denotes the differencing operation applied d times to make the series stationary
- $\Phi(L) = 1 - \phi_1 L - \phi_2 L^2 - \cdots - \phi_p L^p$ is the autoregressive (AR) polynomial
- $\Theta(L) = 1 + \theta_1 L + \theta_2 L^2 + \cdots + \theta_q L^q$ is the moving average (MA) polynomial
- ε_t is a white noise error term

In our implementation, the ARIMA model is applied to the target variable derived from a sliding window of stock prices, with a fixed window size of 5. The model is trained on the training portion of this sequence without exogenous variables. A grid search over combinations of p, d, and q is used to identify the optimal model configuration. For each parameter set, the ARIMA model is fit to the training series, and forecasts are generated for the test period. The best-performing model is selected based on the resulting prediction accuracy.

Because of its simplicity and capability for tasks involving stationary data, ARIMA was chosen as a baseline for our more advanced models. This model is particularly useful for capturing cyclic behavior and providing interpretable forecasts. Some limitations, however, are ARIMAs assumption of linear relationships and non-stationary data. Previously, in an attempt to forecast the S&P 500, this method showcased proficiency in capturing seasonal behavior and short-term forecasts but failed to adapt to sudden fluctuations in the market[20]. While ARIMA serves as a valuable benchmark, its simplicity and shortcomings when modeling non-linear data underscore the need for more robust models; however, involving it in hybrid architecture may offer improved results. Our ARIMA models hyperparameters (p, d, q), representing autoregressive, differencing, and moving average orders, are optimized using grid search. Specific values for (p, d, q) are tested from 1 to 3 for each parameter, and each combination is evaluated on the test set using Mean Squared Error (MSE). This ensures the selection of the best-performing parameter set for accurate stock market predictions.

B. Bidirectional Long Short-Term Memory (BiLSTMs)

BiLSTMs are a variant of the Long Short-Term Memory (LSTM) networks, designed to capture dependencies in temporal data[21]. While LSTMs, and other basic deep learning models, often process data sequentially in one direction, BiLSTMs handle data in parallel layers: one parses the data in the forward direction while the other in the reverse. This structure allows for the model to grasp the full preceding and succeeding context, making it useful for tasks that require an understanding of temporal data.

Bidirectional Long Short-Term Memory (BiLSTM) networks are an extension of standard LSTM networks, designed to capture both past and future temporal dependencies in sequential data. They consist of two LSTM layers processing the input sequence in opposite directions: one forward (left to right) and one backward (right to left), with their outputs concatenated at each time step. The standard LSTM cell operates by controlling the flow of information through three gating mechanisms: the forget gate, input gate, and output gate. These gates regulate the updates to the cell state C_t and hidden state h_t, allowing the model to retain or discard information over long sequences. The governing equations of a unidirectional LSTM for a given time step t are:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \quad \text{(Forget Gate)} \tag{2}$$
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \quad \text{(Input Gate)} \tag{3}$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \quad \text{(Candidate Cell State)} \tag{4}$$
$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \quad \text{(Cell State Update)} \tag{5}$$
$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \quad \text{(Output Gate)} \tag{6}$$
$$h_t = o_t \odot \tanh(C_t) \quad \text{(Hidden State Output)} \tag{7}$$

Here, x_t is the input at time t, h_{t-1} is the previous hidden state, and σ denotes the sigmoid activation function. The symbol $\odot$ denotes element-wise multiplication. The weight matrices W_f, W_i, W_C, W_o and biases b_f, b_i, b_C, b_o are learned during training. In the Bidirectional LSTM architecture used in this study, two layers of BiLSTMs are employed. The first BiLSTM layer processes the input sequences and returns sequences to be fed into a second BiLSTM layer. Dropout regularization is applied after each recurrent layer to mitigate overfitting. Finally, a fully connected dense layer produces the output prediction. To optimize model performance, hyperparameters such as the number of units in each BiLSTM layer, dropout rates, and optimizer type are tuned using Keras Tuners `RandomSearch`. The input data is reshaped into a 3D tensor format $(\text{samples}, \text{timesteps}, \text{features})$, standardized using `StandardScaler`, and then split into training and testing sets. The BiLSTMs ability to learn from both past and future time steps makes it well-suited for financial time series forecasting, where context from both directions can be informative.

Financial time-series data, notably the stock market, often exhibits dependencies where trends are influenced by past behav-

ior, and future events may provide context on the degree of this impact. On top of processing data in both directions, BiLSTM models are built to handle non-linear dependencies, making them effective in managing complex patterns, market shocks, and trend reversals. Previous studies have found that this LSTM variant outperforms its predecessor in tasks involving time-series data, such as price and financial prediction, with the addition of sentiment scores greatly improving performance [12]. It is also shown to be superior in comparison to other statistical-based models, such as ARIMA and GARCH, and traditional machine learning models such as Support Vector Machines (SVMs) and Decision Trees [13]. BiLSTMs serve as an essential component of this study as they capture both forward and backward dependencies in time-series data, making them particularly effective for modeling stock price movements influenced by complex temporal and contextual relationships. In the future, BiLSTMs could serve as the backbone of hybrid models or as a standalone solution for forecasting problems where rich temporal patterns need to be modeled. By leveraging bidirectional processing, BiLSTMs provide a more comprehensive analysis of temporal dependencies in stock market data, making them a vital tool in modern financial forecasting [16]. Our Bidirectional LSTM model leverages Keras Tuner for hyperparameter optimization to ensure the best model performance. A grid of values for key parameters such as the number of units in each LSTM layer (32, 64, 96, 128), dropout rates (0.1 to 0.5 in increments of 0.1), and optimizers (adam and rmsprop) are tested. The tuner evaluates combinations of these parameters over a maximum of 10 trials, selecting the configuration that minimizes validation loss. This systematic approach guarantees the optimal architecture for capturing bidirectional temporal dependencies in stock data.

C. Convolutional Neural Network (CNNs)

CNNs are a class of deep-learning models designed for feature extraction and pattern recognition [4]. Although designed for computer vision-related tasks, CNNs can be adapted for temporal data by slightly altering the architecture of the model. In the context of regression, our CNN will detect local patterns and relationships in our data, excelling at identifying local, short-term movements in the stock market. Convolutional Neural Networks (CNNs) process input data using convolutional layers that apply filters to extract local patterns or features. In the context of time-series forecasting, CNNs slide these filters over sequential data to identify trends, patterns, or anomalies.

Convolutional Neural Networks (CNNs) are widely used for extracting spatial or temporal features from structured data. In the context of time series forecasting, 1D CNNs apply convolutional filters along the time axis to detect short-term patterns in sequential input windows. Given an input time series window $\mathbf{x} \in \mathbb{R}^{T \times F}$, where T is the number of time steps (window size) and F is the number of features per step, a 1D convolutional layer applies a set of K filters $\{\mathbf{w}^{(k)}\}_{k=1}^{K}$ of size h, sliding over the time axis. The output at time step t for filter k is computed as:

$$z_t^{(k)} = \sigma\left(\sum_{i=0}^{h-1}\sum_{j=1}^{F} w_{i,j}^{(k)} \cdot x_{t+i,j} + b^{(k)}\right) \tag{8}$$

where:

- $x_{t+i,j}$ denotes the input at time $t+i$ and feature j
- $w_{i,j}^{(k)}$ is the weight at position i, j in filter k
- $b^{(k)}$ is the bias term for filter k
- $\sigma(\cdot)$ is the activation function (e.g., ReLU or tanh)

Following the convolution, a pooling operation is applied to reduce the spatial dimension of the feature maps. In max pooling with window size p, the pooled output for each filter is:

$$\hat{z}_t^{(k)} = \max_{i=0,\ldots,p-1} z_{t+i}^{(k)} \tag{9}$$

This operation retains the most significant activation within the local region, which helps improve generalization and reduces computational complexity. The resulting pooled features are passed through a flattening layer and then through one or more dense (fully connected) layers to generate a prediction. In this study, the final output layer is a single neuron predicting the next value in the time series. Model optimization is conducted using Keras Tuner's `RandomSearch`, which tunes the number of filters, kernel size h, activation function σ, dropout rate, number of dense units, and optimizer choice. The input data is normalized and reshaped to the format (N, T, F) where N is the number of samples. This allows the CNN to efficiently learn representations across short time windows of stock market data. The CNN's ability to model local temporal correlations without recurrence makes it a computationally efficient yet powerful choice for financial time series prediction tasks.

Due to its distinct kernel components, CNNs excel at feature extraction, most notably identifying relationships between variables like historical prices, sentiment scores, and macroeconomic indicators. In past projects, CNNs have been used to process time-series data, including stock prices and technical indicators, identifying short-term patterns like trends and resistance levels, and even in conjunction with RNNs and LSTMs to enhance performance [14]. In this study, they complement temporal models like BiLSTMs and Transformers by providing a robust framework for extracting meaningful features from raw inputs. They are ideal for hybrid architectures where local feature extraction is a critical preprocessing step for more complex temporal or attention-based models. By efficiently identifying

localized patterns in stock data, CNNs play a critical role in understanding and predicting short-term market movements, enhancing the overall accuracy and robustness of financial forecasting systems. Our CNN model utilizes Keras Tuner to optimize hyperparameters, including the number of filters (32 to 128), kernel sizes (2, 3, 5), activation functions (relu, tanh), dropout rates (0.1 to 0.5), and dense layer units (32 to 128). Additionally, optimizers (adam, rmsprop) are tested. A randomized search is performed over a maximum of 10 trials to find the best combination of parameters that minimizes validation loss. This approach ensures the model effectively captures localized patterns in stock time-series data for accurate predictions.

D. Transformers

Transformers are designed to process data through their defining attention mechanism[22]. Unlike traditional sequence models, Transformers can capture long-range dependencies in sequences by focusing on the most relevant parts of the input, regardless of their position in the sequence. This capability makes them highly efficient and scalable for tasks involving large datasets or long time horizons.

Originally developed for natural language processing, Transformer architectures have shown strong performance in time series forecasting due to their ability to model long-range dependencies without recurrence. In this study, a customized Transformer model is constructed using stacked self-attention layers and feedforward networks, adapted to the temporal structure of stock data. The central component of the Transformer architecture is the scaled dot-product attention mechanism, which operates on a query matrix Q, a key matrix K, and a value matrix V. Each is derived from the same input $X \in \mathbb{R}^{T\times d}$, where T is the time window and d is the feature or embedding dimension. The attention output is computed as:

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d}}\right)V \tag{10}$$

To enhance the models capacity to capture diverse relationships, the Transformer uses multi-head attention. This allows the model to attend to information from multiple representation subspaces. The multi-head attention is given by:

$$\text{MultiHead}(X) = \text{Concat}(\text{head}_1,\ldots,\text{head}_h)W^O \tag{11}$$

where each attention head is defined as:

$$\text{head}_i = \text{Attention}(XW_i^Q, XW_i^K, XW_i^V) \tag{12}$$

Here, W_i^Q, W_i^K, and W_i^V are projection matrices for the i-th head, and W^O projects the concatenated outputs back to the original dimension. The output of the multi-head attention layer undergoes residual connection and layer normalization. This is followed by a feedforward network applied position-wise:

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{13}$$

Dropout is applied after both the attention and feedforward stages to prevent overfitting. In this implementation, the input features are first projected into a higher-dimensional space through a dense embedding layer. The transformer block described above is then applied to this sequence. The resulting tensor is flattened and passed through a dense layer before producing the final scalar output representing the forecasted value. Hyperparameter tuning is conducted using Keras Tuners `RandomSearch` to optimize:

- Embedding dimension (`embed_dim`)
- Number of attention heads (`num_heads`)
- Size of feedforward network (`ff_dim`)
- Dropout rate
- Dense layer units and optimizer type

The Transformer architecture is particularly suited for time series data with potential long-range temporal dependencies, making it a powerful alternative to traditional recurrent models in stock prediction tasks.

Transformers excel at capturing these dependencies efficiently without the need for sequential processing (like RNNs), making them particularly suited for modeling relationships over extended periods (e.g., predicting monthly trends based on yearly data) and handling large, multi-modal datasets, which are common in financial forecasting[6]. While Transformers have been traditionally used for NLP tasks, applications in financial areas have gained traction: used in studies combining historical stock data with textual sentiment analysis from news or social media, integrated to predict stock prices and movements, leveraging their ability to model long-range dependencies in price data, and applied to model interdependencies between assets for dynamic portfolio management[23]. Transformers are likely to remain an anchor in the domain of stock market forecasting, especially with the constant effort toward improvement[15]. Their unparalleled ability to model long-range dependencies and identify contextual relationships positions them well as a candidate for hybrid models, the last foreshadow, and paves the way for accurate and scalable stock market prediction. Our Transformer model uses Keras Tuner to optimize key hyperparameters, including embedding dimension (32 to 128), number of attention heads (2 to 8), feedforward layer size (32 to 128), dropout rate (0.1 to 0.5), and dense layer units (32 to 128). Additionally, optimizers (adam, rmsprop) are explored. Through a randomized search over 10 trials, the best configuration is selected to

minimize validation loss. This hyperparameterization ensures the Transformer effectively captures long-term dependencies and temporal relationships in stock time-series data for accurate forecasting.

E. BiLSTM + CNN + Transformer (Hybrid)

Our Hybrid model is a conjunction of three already powerful architecturesBiLSTMs, CNNs, and Transformers[7]. It aims to leverage their complementary strengths to cancel out each others weaknesses, excelling in handling sequential data, identifying local patterns, and grasping long-term dependencies. Each component, however, plays a specific role. BiLSTMs capture bidirectional temporal dependencies in sequential data, learning from both past and future contexts. CNNs extract local patterns and features, such as short-term trends or anomalies, from time-series data. Transformers handles long-range dependencies and learns contextual relationships with its self-attention mechanism. The Hybrid Model combines the strengths of BiLSTM, CNN, and Transformer architectures to handle both local patterns, temporal dependencies, and long-range relationships in time-series data. Each component contributes uniquely to the model:

To leverage the strengths of multiple deep learning architectures, a hybrid model is constructed by combining convolutional neural networks (CNNs), bidirectional long short-term memory (BiLSTM) networks, transformer-style multi-head attention, and a gradient boosting regressor (GBR) as the final predictor. Each component serves a distinct role in the feature extraction and modeling pipeline for stock price forecasting. The model begins with a 1D convolutional layer that extracts localized temporal patterns from the input sequence. Given an input tensor $X \in \mathbb{R}^{T\times F}$, the convolutional layer applies K filters of size h, yielding:

$$z_t^{(k)} = \sigma\left(\sum_{i=0}^{h-1}\sum_{j=1}^{F} w_{i,j}^{(k)} \cdot x_{t+i,j} + b^{(k)}\right) \tag{14}$$

where σ is a non-linear activation function, and $w_{i,j}^{(k)}$ denotes the learnable parameters of the k-th filter. The convolved output is passed through a bidirectional LSTM layer to capture both forward and backward temporal dependencies. Each LSTM cell computes hidden states using gating mechanisms:

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \quad \text{(Forget Gate)} \tag{15}$$
$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \quad \text{(Input Gate)} \tag{16}$$
$$\tilde{C}_t = \tanh(W_C[h_{t-1}, x_t] + b_C) \quad \text{(Candidate Cell State)} \tag{17}$$
$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \quad \text{(Cell Update)} \tag{18}$$
$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \quad \text{(Output Gate)} \tag{19}$$
$$h_t = o_t \odot \tanh(C_t) \quad \text{(Hidden State)} \tag{20}$$

The BiLSTM output is passed to a transformer-style multi-head attention block. The scaled dot-product attention mechanism is defined as:

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V \tag{21}$$

where Q, K, and V are the query, key, and value matrices obtained from the BiLSTM output. Multiple attention heads attend to information in parallel, and the result is normalized and passed through dense layers. Following flattening and dropout, the final feature vector is extracted from the penultimate dense layer. Instead of producing a direct output, this vector is input to a separate Gradient Boosting Regressor (GBR), which learns a non-linear ensemble of decision trees. Each tree is trained to correct the residual errors of the previous ensemble. The boosting model approximates the true function $f(x)$ by sequentially adding weak learners:

$$f(x) = \sum_{m=1}^{M} \gamma_m h_m(x) \tag{22}$$

where $h_m(x)$ is the m-th regression tree and γ_m is its associated weight. The entire architecture is trained in two phases. First, the deep learning portion is tuned using Keras Tuners `RandomSearch`, optimizing hyperparameters including CNN filter size and activation, LSTM units, attention heads and key dimension, dense units, dropout, and optimizer. After training, the output of the penultimate dense layer is used as input to the GBR, which is trained on these learned representations to make the final prediction. This hybrid approach enables hierarchical feature extraction (via CNN and BiLSTM), global attention modeling (via Transformer), and final non-linear regression (via GBR), providing a robust pipeline for modeling complex financial time series dynamics.

Moreover, this specific combination of architecture is tailored to the limitations and capabilities of each of the model. While BiLSTMs capture short- and mid-term temporal dependencies, they struggle with long-term dependencies and require sequential processing, which can be computationally expensive. CNNs identify localized patterns (e.g., price spikes or momentum shifts) but lacks the ability to model sequential relationships or long-range dependencies. Transformers model long-term dependencies and contextual relationships, refining predictions across extended time horizons but may miss localized or short-term patterns unless guided by additional feature extraction. Previous studies have tested various permutations of this specific architecture[17]. One specific example, CNNs + LSTMs, showed that hybrid models outcompeted their standalone counterparts. Additionally, the integration of attention-blocks (the basis of Transformers) into LSTM systems demonstrated significant improvements involving the inclusion of sentiment and technical indicators. This

model exemplifies the strengths of combining architectures and ensures robust predictions to address the inherent complexity in the issue of stock market prediction. The hybrid model combines CNN, Bidirectional LSTM, Multi-Head Attention, and Gradient Boosting components, with hyperparameter tuning conducted using the Keras Tuner. The tuning process optimizes parameters such as CNN filters (32 to 128), kernel size (2, 3, 5), LSTM units (32 to 128), attention heads (2 to 8), attention key dimensions (32 to 128), dense layer units (32 to 128), dropout rate (0.1 to 0.5), and optimizer choice (adam,rmsprop). A randomized search is performed over 10 trials to minimize validation loss. The hybrid model effectively captures local patterns (CNN), temporal dependencies (LSTM), and global context (Attention) for robust stock price forecasting.

F. Bidirectional Encoder Representations from Transformers (BERT-inspired)

While BERT was initially developed for NLP tasks such as language modeling and sentiment analysis, our altered BERT-inspired model handles time series and numerical data, leveraging BERTs ability to capture context and relationships between elements in a sequence, ideal for this task where stock performance on a certain day affects succeeding days[8]. The BERT-inspired model adapts the Transformer architecture to time-series forecasting. It captures temporal dependencies and relationships in sequential stock data using multi-head attention, feedforward layers, and residual connections.

Bidirectional Encoder Representations from Transformers (BERT) was originally developed for natural language processing. In this study, a BERT-inspired model is adapted to numerical time series data, repurposing its self-attention mechanism and transformer architecture to capture temporal relationships in sliding window sequences. The model takes as input a time series window $X \in \mathbb{R}^{T\times F}$, where T is the number of past time steps and F is the number of features. An initial dense projection maps this input into an embedding space of dimension d, serving as the input for the transformer block. The core of the architecture is the multi-head self-attention mechanism, which computes the attention output over the input embeddings. For a single head, the attention is calculated using:

$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V \tag{23}$$

where $Q = XW^Q$, $K = XW^K$, $V = XW^V$, and $W^Q, W^K, W^V \in \mathbb{R}^{d\times d_k}$ are learned projection matrices. The softmax operation ensures that the attention weights sum to 1 across time steps. To improve the models capacity, multi-head attention applies this computation in parallel across h heads and concatenates the results:

$$\text{MultiHead}(X) = \text{Concat}(\text{head}_1,\ldots,\text{head}_h)W^O \tag{24}$$

The result is passed through dropout and a residual connection with the original embedding, followed by layer normalization:

$$\hat{X}_1 = \text{LayerNorm}(X + \text{Dropout}(\text{MultiHead}(X))) \tag{25}$$

This is followed by a feedforward subnetwork:

$$\hat{X}_2 = \text{LayerNorm}(\hat{X}_1 + \text{Dropout}(W_2 \cdot \text{ReLU}(W_1\hat{X}_1 + b_1) + b_2)) \tag{26}$$

where W_1, W_2 are learned weight matrices of the feedforward layers. Finally, the output across time steps is aggregated using Global Average Pooling, producing a fixed-size feature vector. This is passed through a final dense layer to generate the prediction for the next time step in the series.

Hyperparameters such as embedding dimension, number of attention heads, key dimension, feedforward dimension, dropout rate, and optimizer type are tuned using Keras Tuners `RandomSearch` strategy. Despite BERT's original intent for text, this adaptation demonstrates the architectures flexibility and its potential in modeling complex dependencies in structured numerical time series data.

This BERT-inspired model tuned for time series and numerical data resembles a prototype of a BiLSTM + Transformer Hybrid model. The Bidirectional Encoder, similar to BiLSTMs, captures relationships between past and future data points. The Transformer component, as discussed earlier, is capable of handling extended time horizons, which are crucial for understanding market trends and seasonal behaviors. While the traditional BERT model is extremely popular in financial tasks, such as sentiment analysis and multi-modal forecasting, few studies have implemented a numerical-tailored version of it. By leveraging bidirectional attention and contextual embeddings, the BERT-inspired model provides a novel approach to understanding and predicting stock market behavior, bridging the gap between textual sentiment analysis and numerical forecasting[6]. The BERT-inspired model leverages Keras Tuner to optimize key hyperparameters, including embedding dimension (32 to 128), number of attention heads (2 to 8), key dimension (32 to 128), feedforward layer size (32 to 128), dropout rate (0.1 to 0.5), and optimizer (adam,rmsprop). A randomized search is conducted over 10 trials to identify the best configuration that minimizes validation loss. This hyperparameter tuning ensures the model efficiently captures temporal relationships and contextual dependencies in stock time-series data, leading to robust and accurate predictions.

5 Results and Discussion

In evaluating the performance of predictive models, several metrics are commonly used to quantify accuracy and error.

Mean Absolute Error: The Mean Absolute Error (MAE) measures the average magnitude of errors between the predicted and actual values, where y_i represents the actual values, $\hat{y}_i$ the predicted values, and n the number of observations:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Mean Squared Error: The Mean Squared Error (MSE) emphasizes larger errors by squaring the differences and is particularly sensitive to outliers:

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

Mean Absolute Percentage Error: The Mean Absolute Percentage Error (MAPE) expresses the error as a percentage of the actual values and is useful for understanding relative errors:

$$\text{MAPE} = \frac{100}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right|$$

Coefficient of Determination (R Squared): Finally, the Coefficient of Determination (R^2) assesses the proportion of variance in the actual values explained by the model. Higher R^2 values indicate better model fit, with a maximum value of 1 signifying a perfect prediction:

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

To visualize model performance, several graphs are used for detailed evaluation. The Actual vs. Predicted Values Plot compares the models predictions against the actual values on a scatter plot, ideally forming a straight diagonal line if the predictions are perfect. Deviations from this line indicate prediction errors. The residual plot visualizes the difference between the actual and predicted values against the predicted values. A well-performing model exhibits residuals scattered randomly around zero, indicating no systematic bias. Lastly, the Error Distribution Plot provides a histogram of residuals, showing the frequency and spread of errors. A normal distribution centered around zero suggests that errors are unbiased and randomly distributed, reflecting good model behavior. These visualizations collectively provide insights into the models accuracy, error patterns, and potential areas for improvement.

Table 1 Performance Comparison of All Models

Model	MAE	MSE	MAPE	R^2
ARIMA	4.9881	34.2816	11.0086	0.6046
BiLSTM	1.4128	17.3372	0.0220	0.9873
CNN	2.3909	28.5159	0.0315	0.9790
Transformer	1.8109	19.1469	0.0314	0.9859
Hybrid	4.6626	47.1668	0.0918	0.9653
BERT	2.3690	23.3234	0.0418	0.9829

ARIMA Results

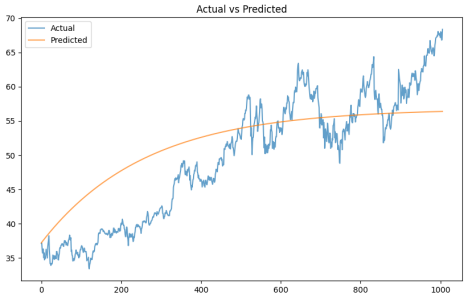

Fig. 1 *
(a) Actual vs. Predicted: X-axis = time (days); Y-axis = closing price.

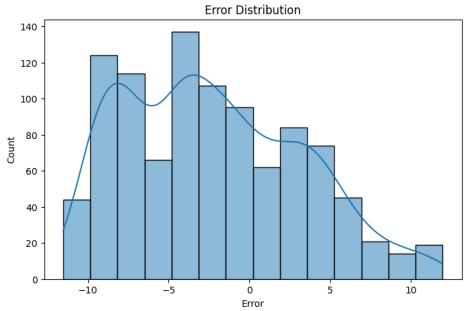

Fig. 2 *
(b) Error Distribution: X-axis = error; Y-axis = frequency.

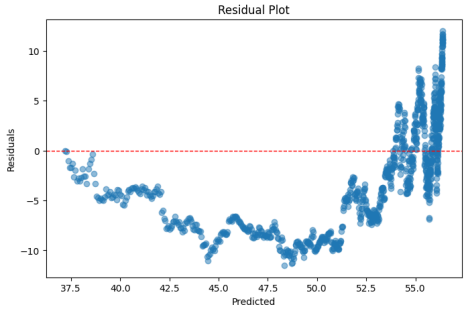

Fig. 3 *
(c) Residual Plot: X-axis = predicted price; Y-axis = residual (actual - predicted).

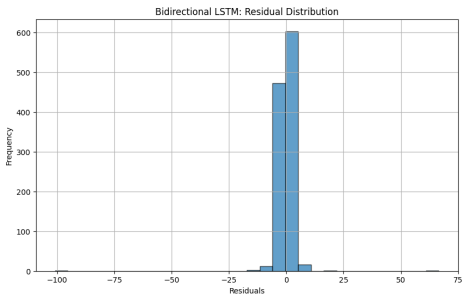

Fig. 4 *
(e) Error Distribution: X-axis = error; Y-axis = frequency.

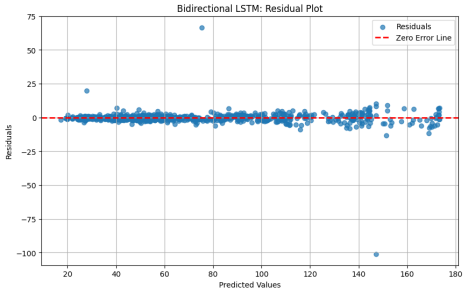

Fig. 5 *
(f) Residual Plot: X-axis = predicted price; Y-axis = residual (actual - predicted).

CNN Results

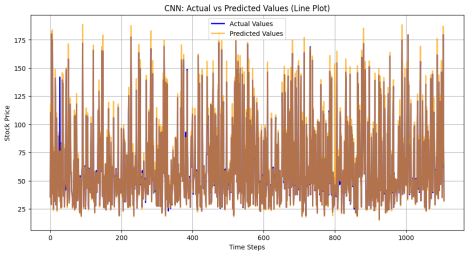

Fig. 6 *
(a) Actual vs. Predicted: X-axis = time (days); Y-axis = closing price.

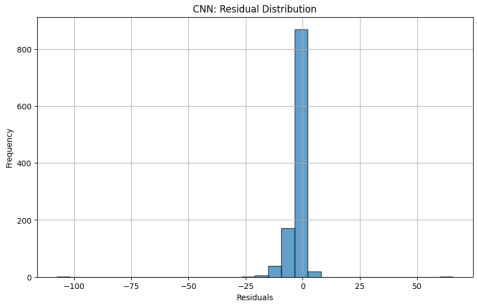

Fig. 7 *
(b) Error Distribution: X-axis = error; Y-axis = frequency.

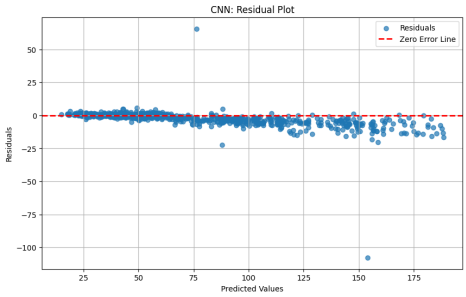

Fig. 8 *
(c) Residual Plot: X-axis = predicted price; Y-axis = residual (actual - predicted).

Transformer Results

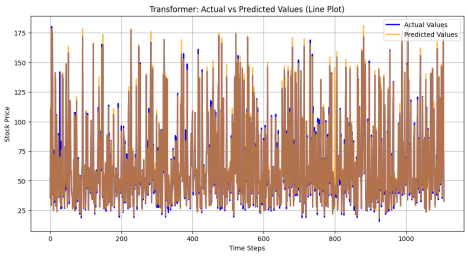

Fig. 9 *
(d) Actual vs. Predicted: X-axis = time (days); Y-axis = closing price.

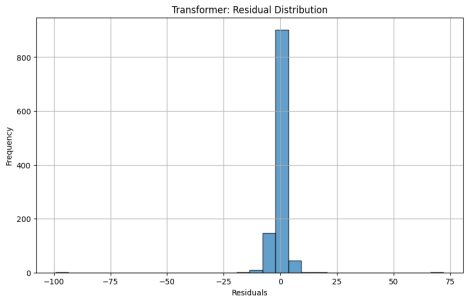

Fig. 10 *
(e) Error Distribution: X-axis = error; Y-axis = frequency.

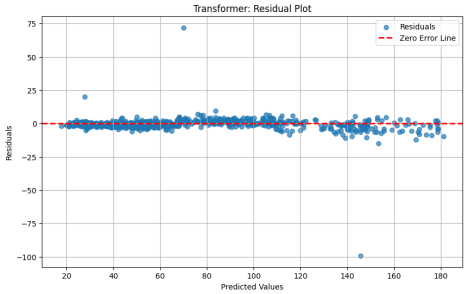

Fig. 11 *
(f) Residual Plot: X-axis = predicted price; Y-axis = residual (actual - predicted).

Hybrid Results

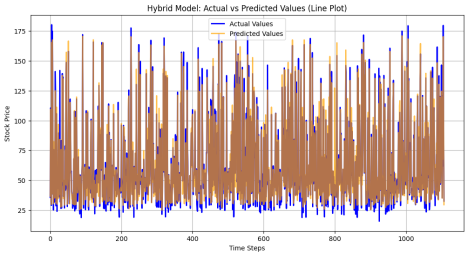

Fig. 12 *
(a) Actual vs. Predicted: X-axis = time (days); Y-axis = closing price.

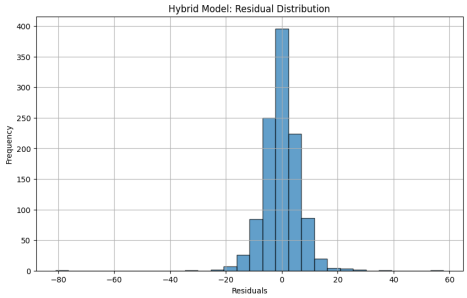

Fig. 13 *
(b) Error Distribution: X-axis = error; Y-axis = frequency.

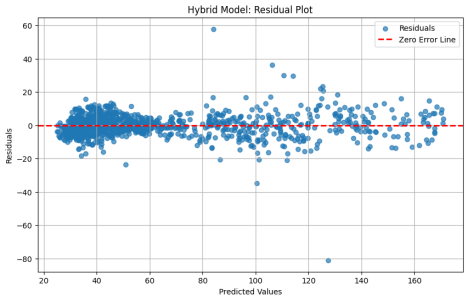

Fig. 14 *
(c) Residual Plot: X-axis = predicted price; Y-axis = residual (actual - predicted).

BERT-Inspired Results

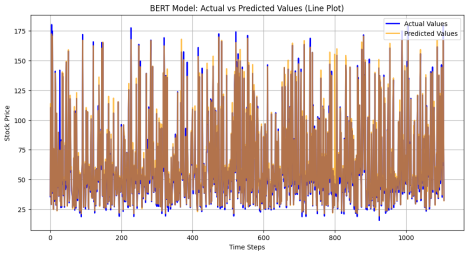

Fig. 15 *
(d) Actual vs. Predicted: X-axis = time (days); Y-axis = closing price.

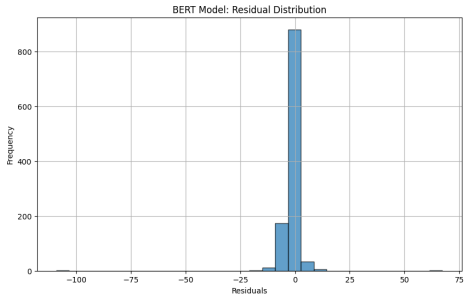

Fig. 16 *
(e) Error Distribution: X-axis = error; Y-axis = frequency.

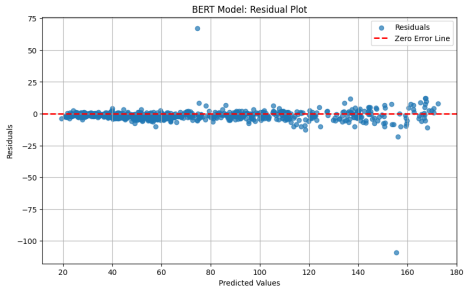

Fig. 17 *
(f) Residual Plot: X-axis = predicted price; Y-axis = residual (actual - predicted).

Best Parameters Summary

ARIMA: $(p,d,q) = (2,0,1)$

Bidirectional LSTM: {'units_1': 64, 'dropout_rate_1': 0.5, 'units_2': 128, 'dropout_rate_2': 0.1, 'optimizer': 'adam'}

CNN: {'filters': 96, 'kernel_size': 2, 'activation': 'relu', 'dropout_rate': 0.1, 'units': 96, 'optimizer': 'adam'}

Transformer: {'embed_dim': 128, 'num_heads': 2, 'ff_dim': 96, 'dropout_rate': 0.2, 'dense_units': 128, 'optimizer': 'rmsprop'}

Hybrid: {'cnn_filters': 96, 'cnn_kernel_size': 3, 'cnn_activation': 'tanh', 'lstm_units': 64, 'attention_heads': 2, 'attention_key_dim': 96, 'dense_units': 64, 'dropout_rate': 0.1, 'optimizer': 'rmsprop'}

BERT-Inspired: {'embed_dim': 32, 'num_heads': 6, 'key_dim': 96, 'ff_dim': 64, 'dropout_rate': 0.1, 'optimizer': 'adam'}

Model Performance Comparison

The experimental results highlight the effectiveness of various deep-learning architectures for stock price prediction. A comparison of key evaluation metricsMean Absolute Error (MAE), Mean Squared Error (MSE), Mean Absolute Percentage Error (MAPE), and R scoredemonstrates that models incorporating sentiment data generally outperform those based solely on numerical features. The ARIMA model serves as a traditional statistical benchmark. While it provides a reasonable R score (0.6046), its relatively high MAE (4.9881) and MSE (34.2816) indicate limitations in capturing the complex patterns within stock price fluctuations. The Bidirectional LSTM model exhibits significant improvements over ARIMA, particularly in terms of R, which reaches up to 0.9973. Notably, when incorporating sentiment data, the MAE is reduced further (1.4128 in the best trial), highlighting the benefit of additional textual information in predicting stock movements.The CNN model achieves strong performance, particularly when sentiment data is integrated. In its best trial, it achieved an MAE of 1.3921 and an R score of 0.9876. The improvement in MAPE when including sentiment data suggests that CNNs effectively capture short-term dependencies, making them well-suited for this task. Transformers consistently deliver high R scores (0.9978 in numerical trials). However, sentiment data does not drastically improve its performance, suggesting that while transformers excel at sequence modeling, additional textual information does not contribute as significantly compared to LSTMs and CNNs. The hybrid model underperforms relative to individual architectures, with MAE values around 4.0867 to 6.3640. This suggests that the added complexity does not necessarily translate into better predictive performance, potentially due to increased sensitivity to hyperparameters or data volume constraints. BERT-based models yield mixed results. The numerical-only version exhibits high variability, with R scores fluctuating between 0.9864 and 0.9919. However, the inclusion of sentiment data significantly stabilizes performance, achieving an MAE as low as 1.9350 with an R of 0.9840.

Hybrid Model Abalation Study

Table 2 Ablation Study Results for the Hybrid Model

Configuration	MAE	MSE	MAPE	R^2
Full Hybrid	1.1390	9.0254	0.0263	0.9876
No CNN	1.1643	5.1989	0.0260	0.9929
No BiLSTM	1.3288	13.2550	0.0317	0.9818
No Attention	1.0824	5.5148	0.0234	0.9924
No GBR	2.7040	13.5606	0.0678	0.9814

To better understand the individual contributions of each architectural component in the hybrid model, we conducted an ablation study by systematically removing one module at a timeCNN, BiLSTM, Attention, and Gradient Boosting Regressor (GBR)and evaluating the performance using standard metrics. The full hybrid model achieved strong overall performance with an MAE of 1.14, MSE of 9.03, MAPE of 2.63%, and R of 0.9876. Interestingly, removing the CNN slightly improved performance in terms of MSE and R, reaching a peak R of 0.9929, which suggests that in this dataset configuration, convolutional feature extraction may have introduced unnecessary complexity or noise. However, excluding the BiLSTM module caused a noticeable decline in performance, with MAE rising to 1.33 and R dropping to 0.9818, confirming the BiLSTMs importance in capturing sequential dependencies. The Attention mechanism, when removed, resulted in slightly better MAE and MAPE scores, indicating that while it can enhance performance in some cases, it may not be essential when temporal dynamics are already well captured by the LSTM. Most notably, removing the GBR and relying solely on the neural networks output led to a substantial performance degradation, with the MAE increasing to 2.70 and MAPE to 6.78%, highlighting the critical role of the gradient boosting layer in refining the final predictions. Overall, the ablation study demonstrates that while certain components like CNN and Attention may offer marginal or context-dependent benefits, BiLSTM and GBR are integral to achieving optimal accuracy and robustness in the hybrid model architecture.

6 Conclusion

This study compared multiple modeling approachesranging from ARIMA to advanced neural architecturesfor forecasting stock prices. The BiLSTM and CNN models, especially when enhanced with sentiment data, consistently outperformed traditional methods in all evaluation metrics. Transformer-based and BERT-inspired models also performed well, though their gains were often incremental relative to their added complexity. The Hybrid model, despite theoretical advantages, did not exceed the performance of simpler models. The results support the growing consensus that deep learning models, when combined with relevant alternative data sources, can significantly enhance financial forecasting. They also show that simpler, well-tuned models can be more effective than overly complex architectures, especially in real-world settings where interpretability and efficiency matter. This study met its objective of benchmarking a variety of models across a unified dataset and methodology, clarifying the trade-offs between complexity, performance, and data usage in stock prediction tasks. Future research could explore additional ensemble techniques, more granular sentiment features, or larger cross-company datasets to validate broader applicability. Testing these models in live or real-time trading scenarios would also provide practical insight into their robustness. The dataset included a limited selection of companies, and while models

were tuned using Keras Tuner, potential overfitting remains a concern for more complex models. Furthermore, all evaluations were based on historical data, and real-time predictive utility was not tested. This research confirms that deep learning modelsespecially BiLSTM and CNNoffer substantial predictive power when thoughtfully applied to structured and sentiment-enriched stock data, pointing toward a more data-driven future for financial decision-making.

References

1 S. Selvin, R. Vinayakumar, E. A. Gopalakrishnan, V. K. Menon and K. P. Soman, *Stock price prediction using LSTM, RNN and CNN-sliding window model*, 2017.

2 G. E. P. Box, G. M. Jenkins, G. C. Reinsel and G. M. Ljung, *Time Series Analysis: Forecasting and Control*, 1978.

3 G. P. Zhang, *Time Series Forecasting Using a Hybrid ARIMA and Neural Network Model*, 2003.

4 S. Bai, J. Z. Kolter and V. Koltun, *An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling*, 2018.

5 A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, ukasz Kaiser and I. Polosukhin, *Attention Is All You Need*, 2017.

6 G. Zerveas, S. Jayaraman, D. Patel, A. Bhamidipaty, C. Tan and C. Eickhoff, *A Transformer-Based Framework for Multivariate Time Series Representation Learning*, 2020.

7 A. Bagde, *Predicting Stock Market Time-Series Data Using CNN-LSTM Neural Network Model*, 2023.

8 J. Devlin, M.-W. Chang, K. Lee and K. Toutanova, *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, 2019.

9 D. Araci, *FinBERT: Financial Sentiment Analysis with Pre-trained Language Models*, 2019.

10 World Bank, *World Bank Open Data*, 2024, `https://data.worldbank.org/`.

11 Google, *Google Trends*, 2024, `https://trends.google.com/trends/`.

12 S. A. Siami-Namini, N. Tavakoli and A. S. Namin, *The Performance of LSTM and BiLSTM in Forecasting Time Series*, 2019.

13 T. Fischer and C. Krauss, *Deep learning with long short-term memory networks for financial market predictions*, 2018.

14 A. Borovykh, S. Bohte and C. W. Oosterlee, *Conditional time series forecasting with convolutional neural networks*, 2018.

15 R. Farsani and E. Pazouki, *A Transformer Self-Attention Model for Time Series Forecasting*, 2021.

16 M. E. Karim, M. Foysal and S. Das, *Stock Price Prediction Using Bi-LSTM and GRU-Based Hybrid Deep Learning Approach*, 2022.

17 W. Bao, J. Yue and Y. Rao, *A deep learning framework for financial time series using stacked autoencoders and long-short term memory*, 2017.

18 J. Arvidsson, *FAANG/MAAMA stock prices OHLCV (daily updated)*, 2024, `https://www.kaggle.com/datasets/joebeachcapital/faangmaama-stock-prices-ohlcv-daily-updated/data`.

19 I. Suchkov, *twitter-sentiment-stock*, 2024, `https://huggingface.co/datasets/suchkow/twitter-sentiment-stock`.

20 P. F. Pai and C. S. Lin, *A hybrid ARIMA and support vector machines model in stock price forecasting*, 2005.

21 A. Graves and J. Schmidhuber, *Framewise phoneme classification with bidirectional LSTM and other neural network architectures*, 2005.

22 J. Bollen, H. Mao and X. Zeng, *Twitter mood predicts the stock market*, 2011.

23 L. Zhang, C. Aggarwal and G.-J. Qi, *Stock Price Prediction via Discovering Multi-Frequency Trading Patterns*, 2017.

24 T. Preis, H. S. Moat and H. E. Stanley, *Quantifying Trading Behavior in Financial Markets Using Google Trends*, 2013.

25 E. F. Fama, *Efficient capital markets: A review of theory and empirical work*, 1970.