The First International Conference on
Machine Learning and Knowledge Discovery (MLKD 2024)
Amirkabir University of Technology, December 18-19, 2024

The Application and Effectiveness of Machine Learning and Deep Learning Methods in Analyzing and Predicting the Shanghai Stock Index

Raziye Rafibakhsh* Ali Mohades Khorasani† MohammadAmin Rezazadeh‡

Abstract

In recent years, stock price prediction has become a pivotal area of research in finance and economics, attracting significant attention because of its potential for generating profits and managing risk. Financial markets, characterized by high volatility, nonlinear dynamics, and complex endogenous patterns, present considerable challenges for investors and analysts. The Shanghai Stock Exchange, one of the largest and most dynamic markets in Asia, has garnered significant interest from researchers and market participants alike because of its sensitivity to economic, political, and social factors.

This article evaluates and compares the performance of traditional time series models, including the Random Walk model, which is considered a key benchmark and is aligned with the Efficient Market Hypothesis, alongside modern machine learning and deep learning techniques for predicting stock prices in the Shanghai Stock Exchange. The impact of data preprocessing techniques and feature selection on the accuracy of these models is also examined. Additionally, an innovative hybrid model, ARMA-CNN-BiLSTM, is proposed, which combines the classical ARMA model with advanced neural networks such as CNN and BiLSTM. This hybrid approach enhances the extraction and analysis of temporal and spatial patterns from financial data, yielding better results compared to the CNN-BiLSTM model. Since the Random Walk model is used as a benchmark in this study, the more the implemented models outperform this benchmark, the more the Efficient Market Hypothesis, which suggests that market prices fully and instantly reflect all available information, will be challenged.

Keywords: stock market price prediction, machine learning, deep learning

1 Introduction

Stock price prediction has emerged as a pivotal area of research in finance, offering significant opportunities for profit generation and risk management in highly volatile markets. The Shanghai Stock Exchange, one of Asia's largest and most dynamic markets, has garnered substantial attention from both academics and professionals due to its sensitivity to various economic, political, and social factors [1, 2].

Traditional time series models, such as the Random Walk and ARIMA, have been extensively used for analyzing stock price movements. The Random Walk model, which is grounded in the efficient market hypothesis, suggests that stock prices follow an unpredictable and random path, rendering future price movements difficult to predict [1]. However, recent advancements in machine learning have uncovered hidden patterns in financial data, providing a more sophisticated means of predicting stock prices that outperforms traditional approaches [2].

The ARIMA model, a well-known linear approach in time series forecasting, effectively captures autoregressive and moving average components but struggles with nonlinear and complex financial data [3, 4]. To address this, modern machine learning techniques such as XGBoost have demonstrated superior performance in stock price prediction, especially in markets like China, where data is extensive and dynamic [6, 7].

Moreover, hybrid models that integrate Convolutional Neural Networks (CNN) and Long Short-Term Memory (LSTM) networks, like CNN-LSTM and CNN-BiLSTM, have shown significant success in forecasting stock prices. By combining spatial feature extraction capabilities with the temporal modeling strengths of LSTM, these hybrid models effectively analyze the complex patterns in financial time series [8, 9, 10].

This paper aims to evaluate and compare the performance of these methods in predicting stock prices in the Shanghai Stock Exchange, assessing the influence of data preprocessing techniques and feature selection on predictive accuracy.

2 Methodology

This section outlines the methods and techniques employed for forecasting the Shanghai Stock Exchange (SSE) index. The primary focus of this study is to evaluate the application and effectiveness of advanced machine learning and deep learning models in analyzing

*Department of Mathematics and Computer Science, Amirkabir University of Technology, raz.rf@aut.ac.ir
†Department of Mathematics and Computer Science, Amirkabir University of Technology, mohades@aut.ac.ir
‡Department of Mathematics and Computer Science, Amirkabir University of Technology, aminrezazadeh21@gmail.com

and predicting the index.

2.1 Data Description

The data used in this study consists of the Shanghai Stock Exchange Index from January 4, 2010, to January 23, 2020, obtained from the *Yahoo Finance* website. The dataset includes daily information such as open, close, high, and low prices, as well as trading volume. Additionally, macroeconomic features such as the Consumer Price Index (CPI) and interest rates were retrieved from the *FRED* database and incorporated as auxiliary data. The data was initially divided into three sets: training (70%), validation (10%), and testing (20%). Several preprocessing steps were then performed, including handling missing values and identifying/removing outliers using Z-score analysis (Figures 2 and 3). The stationarity of the time series was assessed using the Augmented Dickey-Fuller (ADF) and Kwiatkowski-Phillips-Schmidt-Shin (KPSS) tests. Differencing was applied to certain features to achieve stationarity, as this study found that making some features stationary empirically improved model accuracy. Additionally, normalization techniques such as Min-Max scaling and Yeo-Johnson transformation were employed to standardize the data and prepare it for modeling. The statistical analysis of the Shanghai stock market, as displayed in Figures 1, 2, and 3, reveals significant volatility and random fluctuations in stock prices. Analyzing adjusted closing prices during the specified period, we found a mean of \$2801.28 and a standard deviation of \$529.82, indicating high market volatility. Skewness and kurtosis values of \$0.748 and \$1.52, respectively, highlight asymmetric price movements and the occurrence of rare events. These findings are crucial for selecting and refining predictive models to better account for irregular market behaviors, potentially suggesting the use of nonlinear models. Figure 4 further illustrates the overall trend and fluctuations in the market through a time series plot of these prices.

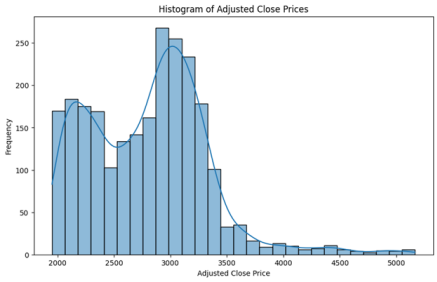

Figure 1: Histogram of adjusted closing prices for the Shanghai stock market

Figure 2: Box plot of adjusted closing prices for the Shanghai stock market before removing outliers

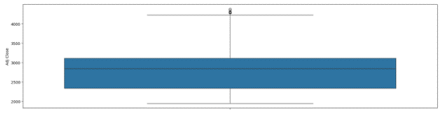

Figure 3: Box plot of adjusted closing prices after removing outliers using Z-score analysis

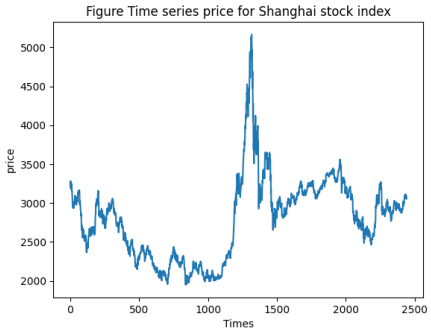

Figure 4: Time series of adjusted closing prices for the Shanghai Stock Index

Statistical tests, including the Phillips-Perron test and the variance ratio test, revealed that the adjusted closing prices follow a random walk. This makes predictions more challenging, as market returns exhibit random fluctuations. The results from the ACF and PACF plots before and after differencing (Figures 5 and 6) demonstrated that, after first differencing, the time series became stationary, making it suitable for modeling with techniques such as ARIMA.

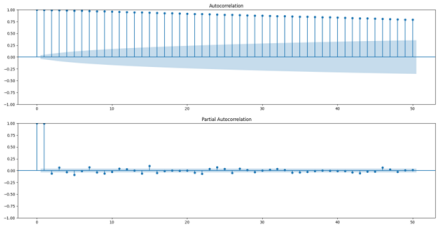

Figure 5: Autocorrelation and partial autocorrelation plots of adjusted closing prices of the Shanghai stock market

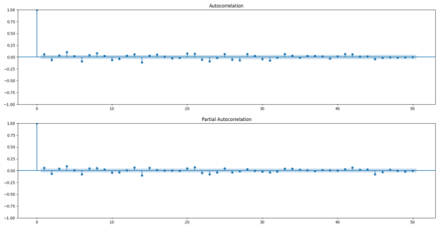

Figure 6: Autocorrelation and partial autocorrelation plots of adjusted closing prices after first differencing

Seasonal and weekly analyses were also conducted to examine temporal effects on the data. The results of the t-test indicated that none of the weekdays had a significant impact on price changes. However, significant changes were observed during the second and third quarters of the year, indicating a notable seasonal effect. Finally, the decomposition of the adjusted closing prices into trend, seasonal, and residual components (Figure 7) revealed that the Shanghai Stock Market exhibits an overall upward trend, with seasonal fluctuations playing a significant role in market changes.

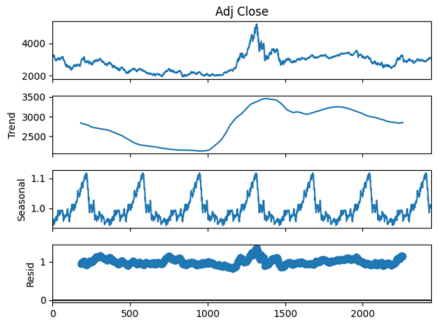

Figure 7: Decomposition of the time series of adjusted closing prices

2.1.1 Min-Max Scaling

Min-Max Scaling is a simple yet effective normalization technique where the data is scaled to a fixed range, typically [0, 1]. This method preserves the relationships between data points (maintaining the relative distribution of the data) while standardizing the range. However, the presence of outliers can affect the scaling process, as they may skew the minimum and maximum values.

The formula for Min-Max scaling is as follows:

$$X' = \frac{X - X_{\min}}{X_{\max} - X_{\min}} \tag{1}$$

Where:

- X is the original data value.
- X' is the scaled data value.
- $X_{\min}$ and $X_{\max}$ are the minimum and maximum values of the original data, respectively.

2.1.2 Yeo-Johnson Transformation

The Yeo-Johnson transformation is a statistical technique used to stabilize variance and make data more normally distributed, similar to the Box-Cox transformation. Unlike Box-Cox, the Yeo-Johnson transformation can be applied to both positive and negative values, making it more flexible. This transformation is defined as follows:

$$y(\lambda) = \begin{cases} \frac{[(y+1)^{\lambda}-1]}{\lambda} & \text{if } \lambda \neq 0 \text{ and } y \geq 0 \\ \ln(y+1) & \text{if } \lambda = 0 \text{ and } y \geq 0 \\ -\frac{[(|y|+1)^{2-\lambda}-1]}{2-\lambda} & \text{if } \lambda \neq 2 \text{ and } y < 0 \\ -\ln(|y|+1) & \text{if } \lambda = 2 \text{ and } y < 0 \end{cases} \tag{2}$$

Where:

- y is the original data value.
- λ is the transformation parameter.

By applying this transformation, data scientists can enhance the robustness and performance of their machine learning models. [**?**]

Both transformations play crucial roles in the preprocessing stage for machine learning models:

- The Box-Cox transformation (and its generalized version, Yeo-Johnson) is beneficial when dealing with non-normal distributions and heteroscedasticity.
- Min-Max scaling is useful for normalizing data within a specific range, ensuring that all features contribute equally to the model.

Proper execution of these transformations can significantly improve the performance of machine learning algorithms by ensuring that the data is in an optimal format for model training.

3 Implementation of models

This section introduces the machine learning and deep learning models employed for predicting the Shanghai Stock Exchange Index in detail.

3.1 Random Walk Model

The Random Walk model is a fundamental approach for forecasting time series data and asset prices in financial markets. It is based on the assumption that asset prices reflect all available information, making future price movements unpredictable. This concept aligns with the Efficient Market Hypothesis (EMH). The Random Walk model is represented as follows:

$$P_t = P_{t-1} + \epsilon_t \tag{3}$$

where P_t is the asset price at time t, and ϵ_t is a random variable with a mean of zero and constant variance. The model assumes that price changes are driven purely by new, random information, implying no identifiable trend for prediction.

Algorithm 1 Simplified Random Walk

Input: Training and test data
Output: Random walk predictions and metrics
Download and split data into training and test sets (70:30) and remove missing values
Initialize the random walk function
for each data point in train and test sets **do**
    Set 'Random Walk Prediction' as previous prediction + random normal value
end for
Calculate metrics for training and test data
return Random walk predictions and performance metrics

As demonstrated in Algorithm 1, the model predicts future prices based on past data with a random component.

On the Shanghai Stock Exchange data, the model shows poor performance: the Root Mean Square Error (RMSE) is 780.857 for training and 285.443 for testing; the Mean Absolute Error (MAE) is 668.156 for training and 233.462 for testing; the Mean Absolute Percentage Error (MAPE) is 27.7 for training and 8.1 for testing; and the coefficient of determination (R^2) is -0.794 for training and -0.293 for testing.

These results highlight the model's limitations in accurately capturing or predicting price changes, suggesting that relying solely on the EMH might not fully explain price movements in the Shanghai stock market. More advanced machine learning models, such as XGBoost or CNN-BiLSTM, can uncover complex patterns and significantly improve prediction accuracy, revealing the limitations of the Random Walk model in real financial markets.

3.2 Random Forest Model

The Random Forest model is an ensemble learning method that enhances predictive accuracy by combining multiple decision trees. The general formula for this model is as follows:

$$f(x) = \frac{1}{N} \sum_{i=1}^{N} T_i(x) \tag{4}$$

In equation (2), $T_i(x)$ represents the output of the i-th tree, and N is the number of trees. To optimize this model, *BayesSearchCV* was used, and the data was split using the *TimeSeriesSplit* method.

Algorithm 2 Random Forest with Min-Max Scaler and Features

procedure RandomForest_MinMaxScaler_Features
    Input: Data, Features
    Output: Model predictions, performance metrics
    Initialize the MinMaxScaler for 'Adj Close' values
    Apply scaling to 'Adj Close' for training, validation, and test sets without data leakage
    Create 'Adj Close (t+1)' column by shifting the 'Adj Close' forward
    Drop rows with null values from training, validation, and test sets
    Initialize a pipeline with Random Forest Regressor
    Define the hyperparameter space for $n\_estimators$, max_depth, and $max\_features$
    Set up a time series cross-validation strategy using 10 splits
    Use BayesSearchCV for hyperparameter optimization with 100 iterations
    Fit the Random Forest model on the training data using the optimal hyperparameters
    Store the validation predictions in a list by predicting for each time step
    Rescale validation predictions and actual values back to the original scale
    Calculate RMSE for the validation data
    Append validation data to the training data and retrain the model
    Store the test predictions in a list by predicting for each time step
    Rescale test predictions to the original scale and calculate metrics
end procedure

As shown in Algorithm **??**, the Random Forest model has been employed with scaling and optimization techniques to enhance accuracy.

3.3 Xtreme Gradient Boosting Model (XGBoost)

The Xtreme Gradient Boosting (XGBoost) model, an advanced form of gradient boosting, was selected for this study due to its ability to handle non-linear data and effectively manage missing or noisy data. In this research, the model was trained using features such as seasonal moving averages and lagged data. Additionally, the Yeo-Johnson transformation was applied for data normalization.

Algorithm 3 XGBRegressor Prediction Model

1: **procedure** XGBRegressorPrediction_Yeo-Johnson_Features
2: **Input:** Data, Features
3: **Output:** Model predictions, performance metrics
4: Shift 'Adj Close' by -1 for all datasets
5: Drop rows with null values in all datasets
6: Set 'Adj Close (t+1)' as the target variable for all datasets
7: Define the XGBRegressor pipeline with a hyperparameter space
8: Initialize TimeSeriesSplit and BayesSearchCV with the pipeline and hyperparameter space
9: Fit BayesSearchCV on the training data
10: Predict and evaluate on validation and test data
11: Concatenate validation data to the training data and refit the model
12: Predict and evaluate test data
13: Inverse transform Yeo-Johnson and calculate metrics
14: **end procedure**

3.4 CNN-BiLSTM Model

This section presents the Convolutional Neural Network-Bidirectional Long Short-Term Memory (CNN-BiLSTM) model, which has been employed for time series forecasting. The model combines convolutional layers and bidirectional LSTM layers, effectively extracting both local features and temporal dependencies in two directions.

The model's weights are learned during the training process via the *model.fit()* function. The model minimizes errors using the *mean-squared-error* loss function and the *Adam* optimizer with a learning rate of 0.001. Grid search was employed for hyperparameter optimization, determining the best values for the number of filters in CNN layers, the number of neurons in BiLSTM layers, and the dropout rates to minimize validation error. The CNN layers serve as initial layers, capturing local patterns from non-stationary data, while the BiLSTM layers learn the temporal dependencies in both forward and backward directions.

Dropout layers were used to prevent overfitting by randomly deactivating certain nodes during training. The overall goal of the model is to optimize the error function.

The convolutional layer operation is defined as follows:

$$\text{Conv}_{\text{out}} = \text{ReLU}(\text{Conv}_{\text{input}} * \text{Filter} + \text{Bias}) \quad (5)$$

As shown in Equation (1), the convolution operation is performed by the CNN filters, and the output is generated through the ReLU activation function.

The forward-pass operation of the BiLSTM layer is as follows:

$$\mathrm{f}_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \quad (6)$$

$$\mathrm{i}_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \quad (7)$$

$$\mathrm{o}_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \quad (8)$$

$$\mathrm{c}_t = \mathrm{f}_t \circ \mathrm{c}_{t-1} + \mathrm{i}_t \circ \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \quad (9)$$

$$\mathrm{h}_t = \mathrm{o}_t \circ \tanh(\mathrm{c}_t) \quad (10)$$

The backward-pass operation of the BiLSTM layer is defined as follows:

$$\mathrm{f}_t^{\text{rev}} = \sigma(W_f^{\text{rev}} \cdot [h_{t+1}^{\text{rev}}, x_t] + b_f^{\text{rev}}) \quad (11)$$

$$\mathrm{i}_t^{\text{rev}} = \sigma(W_i^{\text{rev}} \cdot [h_{t+1}^{\text{rev}}, x_t] + b_i^{\text{rev}}) \quad (12)$$

$$\mathrm{o}_t^{\text{rev}} = \sigma(W_o^{\text{rev}} \cdot [h_{t+1}^{\text{rev}}, x_t] + b_o^{\text{rev}}) \quad (13)$$

$$\mathrm{c}_t^{\text{rev}} = \mathrm{f}_t^{\text{rev}} \circ \mathrm{c}_{t+1}^{\text{rev}} + \mathrm{i}_t^{\text{rev}} \circ \tanh(W_c^{\text{rev}} \cdot [h_{t+1}^{\text{rev}}, x_t] + b_c^{\text{rev}}) \quad (14)$$

$$\mathrm{h}_t^{\text{rev}} = \mathrm{o}_t^{\text{rev}} \circ \tanh(\mathrm{c}_t^{\text{rev}}) \quad (15)$$

As shown in Equations (2) to (12), both temporal directions in the BiLSTM layer are processed, optimizing the information.

The general algorithm for implementing the CNN-BiLSTM model is as follows:

Algorithm 4 CNN-BiLSTM Model with PSO Optimization

procedure CREATE_DATASET(*dataset*, *look_back*=5)
  Input: Data
  Output: Model predictions, performance metrics
  for *each dataset element in range* **do**
    Extract sliding window sequences for look-back period
    Append sequences to dataX and dataY
  end for
  return *dataX*, *dataY*
end procedure
procedure RESHAPE_DATA(*dataX*)
  Reshape dataX to 3D array for CNN input
end procedure
procedure BUILD_MODEL
  Initialize CNN-BiLSTM model
  Add Conv1D layers with filters and kernel sizes
  Add MaxPooling and Bidirectional LSTM layers
  Add Dropout layers for regularization
  Add Dense output layers
end procedure
procedure COMPILE_MODEL
  Compile the model using Adam optimizer and MSE loss function
end procedure
procedure TRAIN_MODEL(*trainX*, *trainY*, *valX*, *valY*)
  Train the model for 200 epochs with a batch size of 32
end procedure
procedure OPTIMIZE_WEIGHTS_WITH_PSO
  Define objective function for PSO based on weighted RMSE
  Define constraints (sum of weights = 1)
  Initialize PSO with bounds and optimize weights
end procedure
procedure EVALUATE_MODEL
  Inverse transform Min-Max and calculate metrics
end procedure

As shown in Algorithm 4, the CNN-BiLSTM model is optimized using Particle Swarm Optimization (PSO) for the weights of different layers.

3.5 ARMA-CNN-BiLSTM Hybrid Model

This innovative model leverages a hybrid approach that combines classical and modern methodologies to forecast financial time series:

The Autoregressive Moving Average (ARMA) Model: The ARMA model is a classical time series model that analyzes data by considering both autoregressive (AR) and moving average (MA) components. It is expressed as follows:

$$\begin{aligned} X_t = \phi_1 X_{t-1} + \phi_2 X_{t-2} + \cdots + \phi_p X_{t-p} \\ + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \cdots + \theta_q \varepsilon_{t-q} + \varepsilon_t \end{aligned} \tag{16}$$

Where:

- X_t is the observed value at time t,
- $\phi_1, \ldots, \phi_p$ are the parameters of the AR component (autoregressive part),
- $\theta_1, \ldots, \theta_q$ are the parameters of the MA component (moving average part),
- p is the order of the AR model, representing how many previous values are used to predict the current value,
- q is the order of the MA model, representing how past forecast errors ($\varepsilon_{t-1}, \varepsilon_{t-2}, \ldots$) are used to adjust predictions,
- ε_t is the error term (white noise) at time t.

The AR component captures the influence of past values on the current observation, while the MA component accounts for past forecast errors. The ARMA model is effective in modeling time series that exhibit both autoregressive behavior and moving average characteristics. However, it may struggle to capture complex, nonlinear patterns often present in real-world data.

Key Insight: The residuals from the ARMA model, which represent the portion of the time series that the ARMA model fails to capture, are passed as inputs to the CNN and BiLSTM models. These residuals allow the CNN and BiLSTM to focus on more complex, nonlinear fluctuations in the data that the ARMA model cannot model effectively.

The Convolutional Neural Network (CNN) Model: In this hybrid approach, CNN is employed to process the residuals from the ARMA model. By using convolutional layers, CNN extracts localized and intricate features from the time series data. It identifies nonlinear patterns in the residuals that are difficult to capture through traditional statistical models. The CNN architecture includes Conv1D layers to process one-dimensional time series data and MaxPooling layers to reduce dimensionality, thereby concentrating on the most significant features.

The Bidirectional Long Short-Term Memory (BiLSTM) Model: The BiLSTM model, a type of recurrent neural network, is used to capture long-term dependencies and model time series in both forward and backward directions. This dual-direction approach enhances the model's ability to remember important patterns over time, improving the accuracy of time series predictions. BiLSTM's ability to preserve time-dependent relationships makes it highly effective in capturing both short- and long-term trends in the residuals.

Optimization with Particle Swarm Optimization (PSO): After obtaining predictions from both CNN and BiLSTM models, the Particle Swarm Optimization (PSO) algorithm is used to combine these predictions optimally. PSO searches for the best combination of model weights that minimizes the prediction error, specifically targeting the Root Mean Squared Error (RMSE). This optimization process ensures that the final hybrid model is finely tuned to deliver the most accurate predictions.

Model Innovation:The integration of the ARMA model with CNN and BiLSTM, optimized by the PSO algorithm, presents a novel approach for time series forecasting. The ARMA model captures linear and stationary components, while CNN and BiLSTM handle complex, nonlinear patterns. The optimization step ensures that the hybrid model achieves superior predictive performance, combining the strengths of each individual component to reduce errors and enhance model efficacy. This hybrid methodology is particularly effective in scenarios where traditional models like ARMA fall short in dealing with nonlinearity, as the CNN and BiLSTM components bridge this gap. Through this integration, the model achieves robust and accurate predictions for complex financial time series.

Algorithm 5 ARMA-CNN-BiLSTM Model with PSO Optimization

procedure ARMA_Model(*train_data*, *order*=(p, q))
    Input: train_data, ARMA(p,q) order
    Fit ARMA model on the training data
    return ARMA residuals for CNN-BiLSTM model
end procedure
procedure Create_Dataset(*dataset*, *look_back*=5)
    Input: dataset, look_back (default=5)
    Output: dataX, dataY
    for *i in range(len(dataset) - look_back - 1)* **do**
        Extract sliding window sequences for look-back period
        Append sequences to dataX and dataY
    end for
    return *dataX*, *dataY*
end procedure
procedure Reshape_Data_For_CNN(*dataX*)
    Reshape dataX to 3D array for CNN input
end procedure
procedure Build_CNN_Model
    Initialize CNN model with Conv1D, MaxPooling, Flatten, Dense, Dropout layers
    Compile the CNN model with Adam optimizer and mean squared error loss
end procedure

Algorithm 5 continue

procedure Build_BiLSTM_Model
    Initialize BiLSTM model with Bidirectional LSTM and Dense layers
    Compile the BiLSTM model with Adam optimizer and mean squared error loss
end procedure
procedure Train_Model(*model*, *trainX*, *trainY*, *valX*, *valY*, *epochs*=300, *batch_size*=16)
    Train the model using the provided training and validation datasets
    return trained model and training history
end procedure
procedure Objective_Function(*weights*)
    Combine predictions from CNN and BiLSTM using the given weights
    Calculate RMSE for train, validation, and test datasets
    return total RMSE
end procedure
procedure Constraint(*weights*)
    Ensure that the sum of weights is equal to 1
end procedure
procedure Optimize_Weights_With_PSO
    Define the initial weights and bounds
    Run PSO to optimize the weights based on the objective function and constraints
    return optimal weights
end procedure
procedure Evaluate_Model
    Inverse transform predictions and actual values to the original scale
    Calculate evaluation metrics
end procedure

4 Results

In this section, the results from implementing machine learning and deep learning models for predicting the Shanghai Stock Exchange index are analyzed and evaluated. The performance of each model on the training, validation, and test datasets is summarized using a range of evaluation metrics. These metrics—Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), Mean Absolute Percentage Error (MAPE), and Coefficient of Determination (R^2)—are crucial for assessing the accuracy and reliability of the models. The following subsections provide detailed explanations of these metrics and their significance in performance evaluation.

4.1 Define Metrics

To evaluate the model performance, several commonly used metrics in regression tasks are defined and explained below. These metrics enable us to quantify the

error between predicted and actual values and assess the model's overall fit.

4.1.1 Coefficient of Determination (R^2)

Explanation: The Coefficient of Determination, denoted as R^2, is a statistical measure that explains the proportion of the variance in the dependent variable that is predictable from the independent variables in a regression model. It provides an indication of the goodness-of-fit of the model, with values ranging between 0 and 1. An R^2 close to 1 signifies that a large proportion of the variability in the target variable is explained by the model, while a value close to 0 indicates poor model performance. This metric helps in assessing how well the model generalizes to unseen data.

The formula for R^2 is:

$$R^2 = 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}} \tag{17}$$

Where:

- $SS_{\text{res}} = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$ is the residual sum of squares, representing the error between the observed and predicted values.
- $SS_{\text{tot}} = \sum_{i=1}^{n}(y_i - \overline{y})^2$ is the total sum of squares, indicating the total variability in the actual data.

A higher R^2 value suggests that the model captures more variance in the data, indicating better performance.

4.1.2 Mean Absolute Error (MAE)

Explanation: The Mean Absolute Error (MAE) is a metric that measures the average magnitude of the errors in a set of predictions, without considering their direction. It represents the average absolute difference between the predicted values and the actual values. MAE is useful because it provides a straightforward interpretation of the model's prediction accuracy in the same units as the target variable. A lower MAE value indicates better predictive performance.

The formula for MAE is:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{18}$$

Where:

- y_i is the actual value,
- $\hat{y}_i$ is the predicted value,
- n is the total number of observations.

MAE is particularly useful when it is important to understand the typical size of errors in predictions. It penalizes large errors equally, making it a robust metric for evaluating model performance across a range of datasets.

4.1.3 Mean Absolute Percentage Error (MAPE)

Explanation: The Mean Absolute Percentage Error (MAPE) measures the accuracy of a forecasting method by expressing the prediction error as a percentage of the actual values. This scale-independent metric allows for comparison across datasets of different scales. MAPE is commonly used in fields like finance and economics, where relative error percentages provide intuitive insights into model performance. However, it can be sensitive to small values in the actual data, which may result in large percentage errors.

The formula for MAPE is:

$$\text{MAPE} = \frac{100}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \tag{19}$$

Where:

- y_i is the actual value,
- $\hat{y}_i$ is the predicted value,
- n is the total number of observations.

MAPE offers a clear indication of the average percentage error made by the model, and a lower MAPE value corresponds to a more accurate model.

These metric definitions and explanations form the foundation for evaluating the performance of predictive models. By understanding these metrics, we can effectively assess the strengths and weaknesses of each model applied to the Shanghai Stock Exchange index prediction task.

Table 1: Machine Learning Model Results

Metric	Training	Validation	Test
Random Forest Model			
RMSE	0.057	0.224	0.246
MAE	0.045	0.207	0.188
MAPE	339129655.51%	18.74%	27.79%
R^2	0.97	0.42	0.33
XGBoost Model			
RMSE	0.00063	0.20254	0.12651
MAE	0.119348	0.19876	0.09886
MAPE	0.00442%	1.81424%	0.90656%
R^2	0.42	0.32	0.17

The performance of the Random Forest model utilizing Min-Max scaling and empirical feature set 1, which includes 'High', 't-6', 'Low', 'Close-Seasonal-Moving-Average-3', and 'Volume', was initially evaluated. The model exhibited strong performance on the training data, achieving an R^2 value of 0.97, signifying its ability to capture the majority of the variance within the dataset. However, its performance declined substantially when applied to the validation and test datasets, with R^2 values decreasing to 0.42 and 0.33, respectively.

This decline suggests the presence of overfitting, where the model was overly tailored to the training data. The root mean square error (RMSE) for the test data was 0.246, and the mean absolute error (MAE) was 0.188, indicating error rates that remain within an acceptable range. However,

the MAPE on the test data was relatively high at 27.79%, highlighting the model's relative weakness in accurately predicting actual values.

The XGBoost model, employing the Yeo-Johnson transformation and empirical feature set 5, which includes 'Open-Seasonal-Moving-Average-3', 'High-Seasonal-Moving-Average-3', 'Low-Seasonal-Moving-Average-3', 't-6', 't-5', 't-4', 't-3', 'Volume', 'CPI', and 'Interest Rate', was subsequently evaluated. This model demonstrated superior performance, particularly on the test dataset, achieving a root mean square error (RMSE) of 0.12651 and a mean absolute error (MAE) of 0.09886, indicating relatively low prediction errors.

Furthermore, the mean absolute percentage error (MAPE) was 0.90656%, highlighting the model's outperformance compared to the other models in terms of predictive accuracy.

Table 2: Deep Learning Model Results

Metric	Training	Validation	Test
ARMA-CNN-BiLSTM Hybrid Model			
RMSE	44.1554	25.6075	36.5494
MAE	31.1329	21.2240	26.5560
MAPE	0.0117%	0.0065%	0.0091%
R^2	0.9916	0.9320	0.9726
CNN-BiLSTM Model			
RMSE	63.36	70.66	48.41
MAE	43.77	65.49	36.93
MAPE	1.60%	2.00%	1.26%
R^2	0.98	0.48	0.95

The hybrid ARMA-CNN-BiLSTM model was employed to predict stock index prices. The results obtained from the model during the training, validation, and testing phases demonstrate superior performance compared to other models in this study.

A detailed examination of the results shows that the hybrid model outperforms the CNN-BiLSTM model across all evaluation metrics:

- Root Mean Squared Error (RMSE) in the testing phase for the hybrid model was 36.5494, which is lower compared to 48.41 in the CNN-BiLSTM model.
- Mean Absolute Error (MAE) in the testing phase for the hybrid model was 26.5560, showing a reduction from 36.93 in the CNN-BiLSTM model.
- Mean Absolute Percentage Error (MAPE) for the hybrid model in the testing phase was 0.0091%, significantly lower than 1.26% in the CNN-BiLSTM model.
- Coefficient of Determination (R^2) for the hybrid model in the testing phase was 0.9726, higher than 0.95 in the CNN-BiLSTM model.

These findings underscore the effectiveness of the ARMA-CNN-BiLSTM hybrid model in modeling and forecasting the complex dynamics of the stock market, positioning it as a reliable method for financial forecasting. The substantial improvements in RMSE, MAE, and MAPE, along with an increase in R^2, confirm the significant enhancements introduced by this new architecture. Optimization techniques such as PSO have played a crucial role in improving the accuracy of these models, leading to substantial improvements in the results.

The superior performance of machine learning and hybrid models compared to the Random Walk model suggests that prices do not solely follow random and new information. Instead, more complex patterns are embedded in market data, which advanced models are capable of uncovering. This clearly challenges the Efficient Market Hypothesis (EMH).

4.2 Discussion and Comparison

Table 3: Comparison of Random Forest Model for Test Results

Metric	Test
Random Forest Model with JP Morgan Data Compared to Model in this Article	
RMSE	0.29
MAE	0.21
MAPE	0.59%
R^2	0.99
Random Forest Model with JP Morgan Data	
RMSE	1.41
MAE	-
MAPE	0.93%
R^2	-

In the article cited as [11], the Random Forest model was employed to predict the closing price of JPMorgan stocks. This model relies on historical stock data to forecast the next day's closing price. The dataset used comprises daily stock information spanning ten years, including variables such as High, Low, Open, Close, Adjusted Close, and Trading Volume.

Standard error metrics such as Root Mean Square Error (RMSE) and Mean Absolute Percentage Error (MAPE) were utilized to evaluate performance. The results indicate that the Random Forest model provided relatively accurate predictions for JPMorgan's closing prices. However, a notable improvement in model accuracy has been observed with the Random Forest model implemented in this article.

To enhance prediction accuracy, new variables were derived from these primary data points, including the difference between High and Low prices (H-L), the difference between Open and Close prices (O-C), and moving averages over 7, 14, and 21 days, along with a 7-day standard deviation.

Specifically, the values of RMSE and MAPE in the proposed model have decreased to 1.12 and 0.34, respectively, demonstrating a significant enhancement in performance, as shown in the comparison table.

Table 4: Comparison of XGBoost Model for Test Results

Metric	**Test**
XGBoost Model with Microsoft Data Compared to Model in this Article	
RMSE	0.0049
MAE	0.0045
MAPE	0.5383%
R^2	0.9781
XGBoost Model with Microsoft Data	
RMSE	-
MAE	21.72
MAPE	16.87%
R^2	-

In the study cited by [12], the XGBoost model was employed with historical stock price information for Microsoft Corporation (MSFT). The variables employed in this analysis include the date, lowest price, highest price, opening price, closing price, adjusted closing price, and trading volume. During the preprocessing stage, the data were scrutinized for unwanted values or missing data to ensure their suitability for modeling with XGBoost.

The dataset was divided into training (60%) and testing (40%) subsets. Standard error metrics such as Mean Absolute Error (MAE) and Mean Absolute Percentage Error (MAPE) were used to evaluate the model's performance. Implementing the XGBoost model on this data with the features detailed in Table 4 resulted in MAE and MAPE values of 21.72 and 16.34, respectively.

Table 5: Comparison of CNN-BiLSTM Model for Test Results

Metric	**Test**
CNN-BiLSTM Model with Shanghai Composite Index data used in this article	
RMSE	0.00855
MAE	0.0065
MAPE	91.006%
R^2	0.9681
CNN-BiLSTM Model with Shanghai Composite Index data from other studies	
RMSE	32.065
MAE	22.715
MAPE	-
R^2	0.9681

In the study by Lu et al.[13], the CNN-BiLSTM model was trained and tested on data from the Shanghai Composite Index, consisting of 7,083 trading days. The dataset includes features such as opening price, highest price, lowest price, closing price, volume, and turnover. The model utilized the first 6,083 days for training and the final 1,000 days for testing.

The CNN-BiLSTM-AM model leverages a combination of Convolutional Neural Networks (CNN), Bidirectional Long Short-Term Memory (BiLSTM) networks, and an Attention Mechanism (AM) for stock price prediction. Initially, the stock data is standardized and fed into the input layer. The CNN layer then extracts temporal features, and the pooling layer reduces the dimensionality of these features. The BiLSTM layer captures temporal dependencies by analyzing bidirectional information. The attention mechanism focuses on the relative importance of features at different times, improving the accuracy of predictions by concentrating on key aspects. Finally, the output layer provides the predicted closing stock price. This model demonstrates high accuracy in price prediction through careful training and feature optimization.

To evaluate model performance, standard error metrics such as RMSE, MAE, and R^2 were used. When the same data was applied to the CNN-BiLSTM model with the features outlined in Table 5, and as described earlier in this article, improvements were observed in the RMSE, MAE, and R^2 values.

In future research, it is suggested to explore advanced techniques such as Vector Quantization and Deep Autoencoders, as applied by Zarringhalam et al. [14] for loop closure detection optimization in SLAM systems. Vector Quantization, with its ability to classify and cluster data, can help analyze complex financial features more effectively and uncover hidden patterns in financial data.

Additionally, Deep Autoencoders, with their capability to compress large data sets and detect complex and nonlinear patterns, could play a crucial role in improving prediction accuracy, particularly in volatile markets such as the Shanghai Stock Exchange. These models are highly suitable for identifying intricate relationships between financial variables that traditional models may not capture.

Given the high volatility and complexity of the Shanghai Stock Exchange, employing these techniques either as part of hybrid models or independently could significantly enhance forecasting accuracy. Moreover, integrating these methods with existing models like ARMA-CNN-BiLSTM could help uncover hidden patterns and nonlinear dynamics in financial markets. The utilization of these techniques will not only enrich current models but also open new avenues for improving stock price prediction and risk management strategies.

Future researchers can take significant steps towards enhancing the accuracy and efficiency of financial forecasting models by focusing on these innovative techniques.

Conflicts of Interest

The authors declare that there is no conflict of interest regarding the publication of this article.

References

[1] B. G. Malkiel. The efficient market hypothesis and its critics. *Journal of Economic Perspectives*, 17(1):59–82, 2003.

css Copy code

[2] A. W. Lo and A. C. MacKinlay. Stock market prices do not follow random walks: Evidence from a simple specification test. *The Review of Financial Studies*, 1(1):41–66, 1988.

[3] J. D. Hamilton. *Time Series Analysis*. Princeton University Press, 2020.

[4] A. Pankratz. *Forecasting with Univariate Box-Jenkins Models: Concepts and Cases*. John Wiley & Sons, 2012.

[5] G. Zhang. Time series forecasting using a hybrid ARIMA and neural network model. *Neurocomputing*, 50:159–175, 2003.

[6] T. Chen and C. Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794, 2016.

[7] X. Li, C. Qin, and H. Wan. Gene expression programming for stock market forecasting. *Applied Soft Computing*, 76:136–146, 2019.

[8] I. E. Livieris, G. Pintelas, and P. Pintelas. A CNN–LSTM model for gold price time-series forecasting. *Neural Computing and Applications*, 32(23):17351–17360, 2020.

[9] J. Kim, S. Ahn, and H. Choi. Forecasting stock prices with a hybrid model combining convolutional neural networks and LSTM. *Expert Systems with Applications*, 82:64–76, 2018.

[10] W. Bao, J. Yue, and Y. Rao. A deep learning framework for financial time series using stacked autoencoders and long short-term memory. *PLOS ONE*, 12(7):e0180944, 2017.

[11] M. Vijh, D. Chandola, V. A. Tikkiwal, and A. Kumar. Stock closing price prediction using machine learning techniques. *Procedia Computer Science*, 167:599–606, 2020.

[12] D. S. Kumar, B. C. Thiruvarangan, A. Vishnu, A. S. Devi, and D. Kavitha. Analysis and prediction of stock price using hybridization of SARIMA and XGBoost. In *2022 International Conference on Communication, Computing and Internet of Things (IC3IoT)*, pages 1–4. IEEE, March 2022.

[13] W. Lu, J. Li, J. Wang, and L. Qin. A CNN-BiLSTM-AM method for stock price prediction. *Neural Computing and Applications*, 33(10):4741–4753, May 2021.

[14] M. Zarringhalam, S. Moein, and M. A. Azizi. Deep autoencoder and vector quantization for loop closure detection optimization in SLAM. *IEEE Robotics and Automation Letters*, 7(2):2154–2161, 2022.