MACHINE LEARNING : AN EFFICIENT HYBRID APPROACH FOR FORECASTING REAL-TIME STOCK MARKET INDICES

Report submitted to the SASTRA Deemed to be University

as the requirement for the course

CSE300 - MINI PROJECT

Submitted by

JAI ADITHYA A

(Reg No-126156054, B. Tech Computer science and AIDS)

ANIRUDH SOORIYAMOORTHY

(Reg No- 126156009, B. Tech Computer science and AIDS)

KIRAN SOORYA R S

(Reg No-126003133, B. Tech Computer science)

MAY 2025

SCHOOL OF COMPUTING

THANJAVUR, TAMIL NADU, INDIA – 613401

SASTRA

ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION

DEEMED TO BE UNIVERSITY

(U/S 3 of the UGC Act,1956)

THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

SCHOOL OF COMPUTING

THANJAVUR – 613 401

Bonafide Certificate

This is to certify that the report titled **"An efficient hybrid approach for forecasting real-time stock market indices"** submitted as a requirement for the course, **CSE300-MINI PROJECT** for B.Tech. is a bonafide record of the work done by

Mr. JAI ADITHYA A (Reg. No.: 126156054, B. Tech Computer science and AIDS),

Mr. ANIRUDH.S (Reg. No.: 126156009, B. Tech Computer science and AIDS) and

Mr. KIRAN SOORYA R S (Reg. No.: 126003133, B. Tech Computer science) during the academic year 2024-25, in the School of Computing, under my supervision.

Signature of Project Supervisor :

Name with Affiliation : Dr. PLK PRIYADARSINI, AP-III, SOC

Date : 03.05.2025

Mini Project *Viva voce* held on _____

Examiner 1 **Examiner 2**

ACKNOWLEDGEMENTS

We would like to thank our Honorable Chancellor **Prof. R. Sethuraman** for providing us with an opportunity and the necessary infrastructure for carrying out this project as a part of our curriculum.

We would like to thank our Honorable Vice-Chancellor **Dr. S. Vaidhyasubramaniam** and **Dr. S. Swaminathan**, Dean, Planning & Development, for the encouragement and strategic support at every step of our college life.

We extend our sincere thanks to **Dr. R. Chandramouli**, Registrar, SASTRA Deemed to be University for providing the opportunity to pursue this project.

We extend our heartfelt thanks to **Dr. V. S. Shankar Sriram**, Dean, School of Computing, **Dr. R. Muthaiah,** Associate Dean, Research, **Dr. K.Ramkumar**, Associate Dean, Academics, **Dr. D. Manivannan,** Associate Dean, Infrastructure, **Dr. R. Algeswaran**, Associate Dean, Students Welfare

Our project supervisor **Dr. PLK PRIYADARSINI**, Associate Professor, School of Computing was the driving force behind this whole idea from the start. Her deep insight in the field and invaluable suggestions helped us in making progress throughout our project work. We also thank the project review panel members for their valuable comments and insights which made this project better.

We would like to extend our gratitude to all the teaching and non-teaching faculties of the School of Computing who have either directly or indirectly helped us in the completion of the project.

We gratefully acknowledge all the contributions and encouragement from our families and friends resulting in the successful completion of this project. We thank you all for providing us an opportunity to showcase our skills through this project.

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

ARIMA	Autoregressive Integrated Moving Average
LSTM	Long Short-Term Memory
H.BLSTM	Hybrid Bidirectional Long Short-Term Memory
MAE	Mean Absolute Error
RMSE	Root Mean Square Error
MAPE	Mean Absolute Percentage Error
EMA	Exponential Moving Average
ML	Machine Learning
DL	Deep Learning
SVM	Support Vector Machine
SVR	Support Vector Regression
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
IncLSTM	Incremental Long Short-Term Memory
HFT	High-Frequency Trading

ABSTRACT

The stock market's volatility, noise, and information overload necessitate efficient prediction methods. Forecasting index prices in this environment is complex due to the non-linear and non-stationary nature of time series data generated from the stock market. Machine learning and deep learning have emerged as powerful tools for identifying financial data patterns and generating predictions based on historical trends. However, updating these models in real-time is crucial for accurate predictions. Deep learning models require extensive computational resources and careful hyperparameter optimization, while incremental learning models struggle to balance stability and adaptability. This paper proposes a novel hybrid bidirectional-LSTM (H.BLSTM) model that combines incremental learning and deep learning techniques for real-time index price prediction, addressing these scalability and memory challenges. The method utilizes both univariate time series derived from historical index prices and multivariate time series incorporating technical indicators. Implementation within a real-time trading system demonstrates the method's effectiveness in achieving more accurate price forecasts for major stock indices globally through extensive experimentation. The proposed model achieved an average mean absolute percentage error of 0.001 across nine stock indices, significantly outperforming traditional models. It has an average forecasting delay of 2 s, making it suitable for real-time trading applications.

Keywords: Stock Market Prediction, Real-time Forecasting, Deep Learning, Incremental Learning, H.BLSTM, Technical Indicators

TABLE OF CONTENTS

CHAPTER 1

SUMMARY OF BASE PAPER

Title	:	An Efficient Hybrid Approach for Forecasting Real-Time Stock Market Indices.
Publisher	:	Journal of King Saud University - Computer and Information Sciences
Year	:	2024
Journal	:	ScienceDirect
Indexing	:	SCI / Scopus
Base paper URL	:	https://www.sciencedirect.com/science/article/pii/S1319157824002696?via%3Dihub

1.1 INTRODUCTION

The stock market is one of the most dynamic and volatile domains, driven by numerous interdependent factors including economic indicators, company performance, investor sentiment, and global events. Stock indices, which represent the aggregated performance of selected stocks, are widely used by analysts, investors, and policymakers as a benchmark for market trends and economic health. Forecasting these indices in real-time is an inherently complex task due to the nonlinear, non-stationary, and noisy nature of financial time series data.

Traditional models such as ARIMA and linear regression have shown limited effectiveness in handling such data, particularly in high-frequency trading environments where rapid decision-making is critical. Machine learning (ML) models like Decision Trees, Random Forests, and Support Vector Machines (SVM) offer improved performance but still fall short in adapting to evolving data streams without full retraining. Meanwhile, deep learning (DL) models such as Long Short-Term Memory (LSTM) networks have demonstrated strong performance in modelling sequential data by capturing long-range dependencies and learning intricate patterns. However, they typically require extensive training time and computational resources, and they do not adapt easily to real-time updates without reprocessing entire datasets.

To overcome these limitations, researchers have explored incremental learning techniques, which update model parameters on-the-fly as new data becomes available. Although these techniques are lightweight and responsive, they often sacrifice long-term memory and forecasting accuracy. This motivated the development of hybrid models that aim to combine the strengths of deep learning and incremental learning.

The hybrid Bidirectional LSTM (H.BLSTM) model proposed in the base paper seeks to address these challenges by integrating both approaches. By using bidirectional LSTM layers, the model captures contextual information from both past and future time steps, enhancing its predictive power. At the same time, an incremental learning mechanism allows the model to incorporate new data during trading sessions, improving adaptability and responsiveness. Technical indicators such as EMA are also included to transform univariate index data into multivariate time series, further enhancing model input quality. This synergy makes the H.BLSTM model particularly suitable for real-time stock market forecasting in high-frequency trading environments, where prediction speed and accuracy are paramount.

1.2 RELATED WORK

Numerous studies have investigated various models for stock market forecasting. Early works relied on statistical methods such as ARIMA, which assume linearity and stationarity—assumptions that do not hold in real financial environments. To overcome these limitations, researchers began exploring machine learning models including Decision Trees, Random Forests, and Support Vector Machines (SVM). For instance, Guo et al. proposed an adaptive SVR model for predicting stock prices on different time scales, demonstrating that dynamic parameter tuning significantly improves prediction accuracy.

With the rise of deep learning, models such as LSTM and CNN-LSTM have become popular due to their superior performance in capturing temporal dependencies. Rundo et al. introduced a reinforcement learning model combined with supervised learning to enhance short-term forecasting in the FOREX market. Zhou et al. developed an adversarial training framework using LSTM and CNN for high-frequency trading data, achieving reduced forecast error.

Incremental learning has also gained attention for real-time applications. Qiu et al. presented an ensemble incremental model combining discrete wavelet transform and SVR. Wang et al. introduced IncLSTM, an incremental LSTM framework that blends ensemble learning with transfer learning, significantly enhancing adaptability and reducing training time. Moreover, Shahparast et al. proposed a fuzzy incremental model that reacts quickly to market trends and outperforms batch models in volatile conditions.

However, these models have limitations. Traditional deep learning models are computationally intensive and lack online adaptability. Incremental models, while adaptive, often fail to retain historical patterns effectively. The H.BLSTM model was proposed to address these challenges by integrating deep learning's forecasting power with incremental learning's responsiveness.

1.3 PROBLEM STATEMENT

Financial data is constantly evolving, and traditional static models are not equipped to adapt to new data in real time. This creates a gap in forecasting accuracy and model

responsiveness, especially in high-frequency trading scenarios where prediction delays can impact decision-making and profitability

1.4 OBJECTIVE

The objective of this project is to design and develop a robust, scalable, and efficient hybrid model for real-time forecasting of stock market indices. The model should accurately predict index movements by leveraging both historical and real-time data. It must be capable of adapting to new data through incremental updates, while maintaining the forecasting power of deep learning techniques. By incorporating technical indicators such as the Exponential Moving Average (EMA), the model aims to enhance the information extracted from univariate series, converting them into multivariate series for improved prediction accuracy. Additionally, the model should ensure low-latency forecasting suitable for high-frequency trading environments, with minimal delay and reduced computational cost.

1.5 PROPOSED SOLUTION AND SYSTEM ARCHITECTURE

1.5.1 STUDY AREA

The proposed H.BLSTM model is designed to combine the strengths of bidirectional deep learning with the adaptability of online learning. The system includes several components:

- **Data Collection**: Historical data is sourced using Python web scraping libraries, while real-time data is streamed at fixed intervals (e.g., every 15 minutes).
- **Data Preprocessing**: Includes removing null values, standardizing timestamps, handling outliers, and converting categorical data (if any) using encoding methods.
- **Feature Engineering**: EMA and other statistical indicators are calculated to create multivariate time series from basic price data. Autocorrelation and cross-correlation plots help determine relevant features and lags.
- **Model Development**: The H.BLSTM model consists of stacked Bidirectional LSTM layers followed by dense layers for final output. It processes historical sequences in both directions to enhance prediction accuracy.
- **Incremental Learning**: The model is updated incrementally during each session using online gradient updates, and a full batch retraining is performed at the end of each trading session to stabilize weights.
- **Evaluation Metrics**: The system is evaluated using MAE, RMSE, and MAPE. A comparison of actual and predicted values is visualized for interpretability.

1.5.2 THE DOW JONES INDUSTRIAL AVERAGE (DJI)

S.NO	FEATURES	DESCRIPTION
1	Dataset Name	Dow Jones Industrial Average (DJI)
2	Source	Financial market APIs or platforms such as Yahoo Finance, Alpha Vantage, etc.
3	Data Type	Time Series
4	Interval	15-minute intervals
5	Attributes	Open, High, Low, Close, Volume
6	Number of Records	Varies depending on duration (typically thousands per month)
7	Period Covered	Real-time / Historical (configurable)
8	Domain	Financial Markets / Stock Market Prediction
9	Target Variable	Close Price (used for forecasting)
10	Purpose	Forecast future stock index values using EMD-LSTM hybrid model

Table 1.1 Description Of DJI Dataset

The Dow Jones Industrial Average (DJI) is a prominent stock market index that tracks the performance of 30 major publicly traded companies listed in the United States.The dataset consists of real-time 15-minute interval stock index values, including features such as opening price, highest and lowest prices within the interval, closing price, and trading volume.

1.5.3 Work Flow Diagram

The workflow of the proposed hybrid forecasting model begins with the collection of raw stock market index data. This data is initially preprocessed, including normalization and noise reduction. Next, the Empirical Mode Decomposition (EMD) technique is applied to decompose the time series into a set of Intrinsic Mode Functions (IMFs), each representing different frequency components. These IMFs are then individually fed into separate Long Short-Term Memory (LSTM) models, which are trained to capture the temporal patterns and dependencies. After prediction, the outputs from all LSTM models are aggregated to reconstruct the final forecast of the original stock index. The overall workflow ensures enhanced prediction accuracy by leveraging both decomposition and deep learning capabilities in Fig 1.1

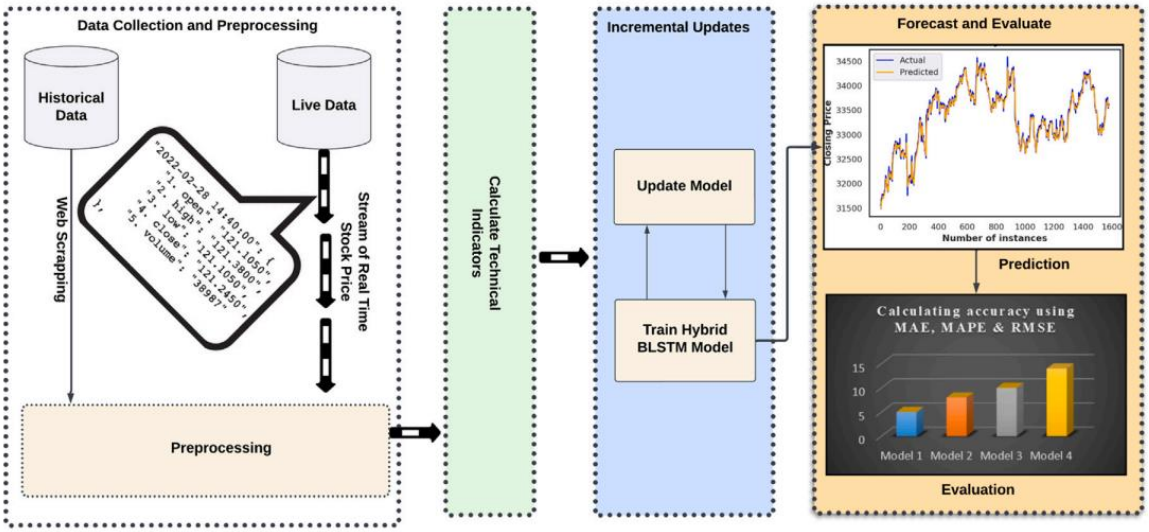


Fig. 2. General overview of the proposed approach.

Fig 1.2.Workflow Diagram

1.6 METHODOLOGY AND IMPLEMENTATION

The methodology proposed in the base paper is designed to enable real-time stock market forecasting using a hybrid approach that integrates deep learning with incremental learning strategies. The implementation of the system follows a systematic and modular workflow as detailed is given in Fig 1.2

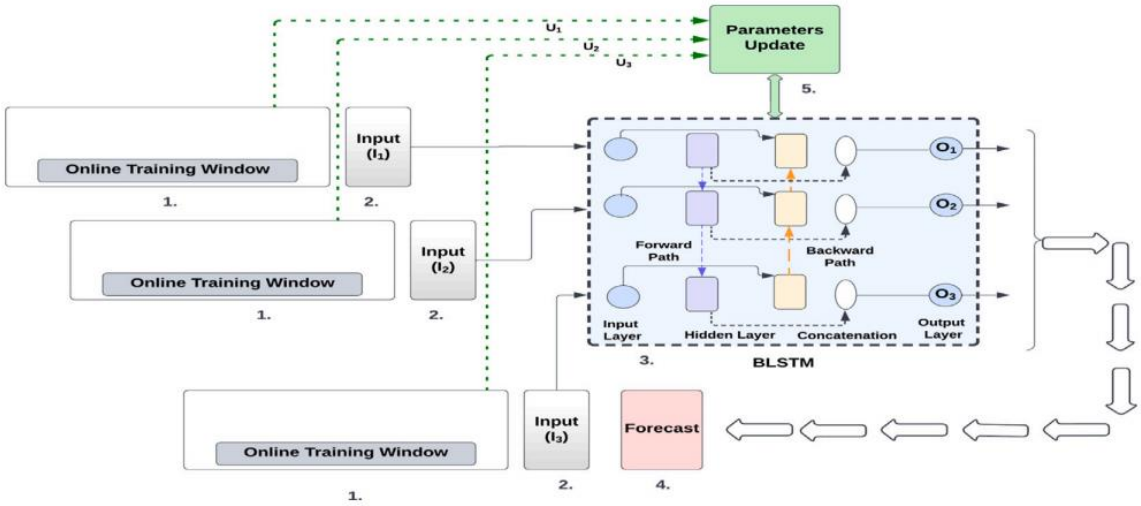


Fig. 3. Architecture of H.BLSTM model.

Fig 1.3. Architecture

1.6.1 MODULE 1: DATA PREPROCESSING

This module involves the acquisition and preprocessing of historical stock market index data. This step is crucial for ensuring the quality and consistency of the input to the forecasting model. The raw data, typically consisting of time-stamped price values, is subjected to normalization to scale the values between 0 and 1. This helps in improving the convergence of deep learning models. Additionally, any missing values or anomalies in the dataset are handled appropriately using imputation or smoothing techniques. The cleaned and normalized data is then structured into sequences to prepare it for the subsequent Empirical Mode Decomposition (EMD) stage. This preprocessing step ensures that the data is stationary and ready for effective decomposition and learning.

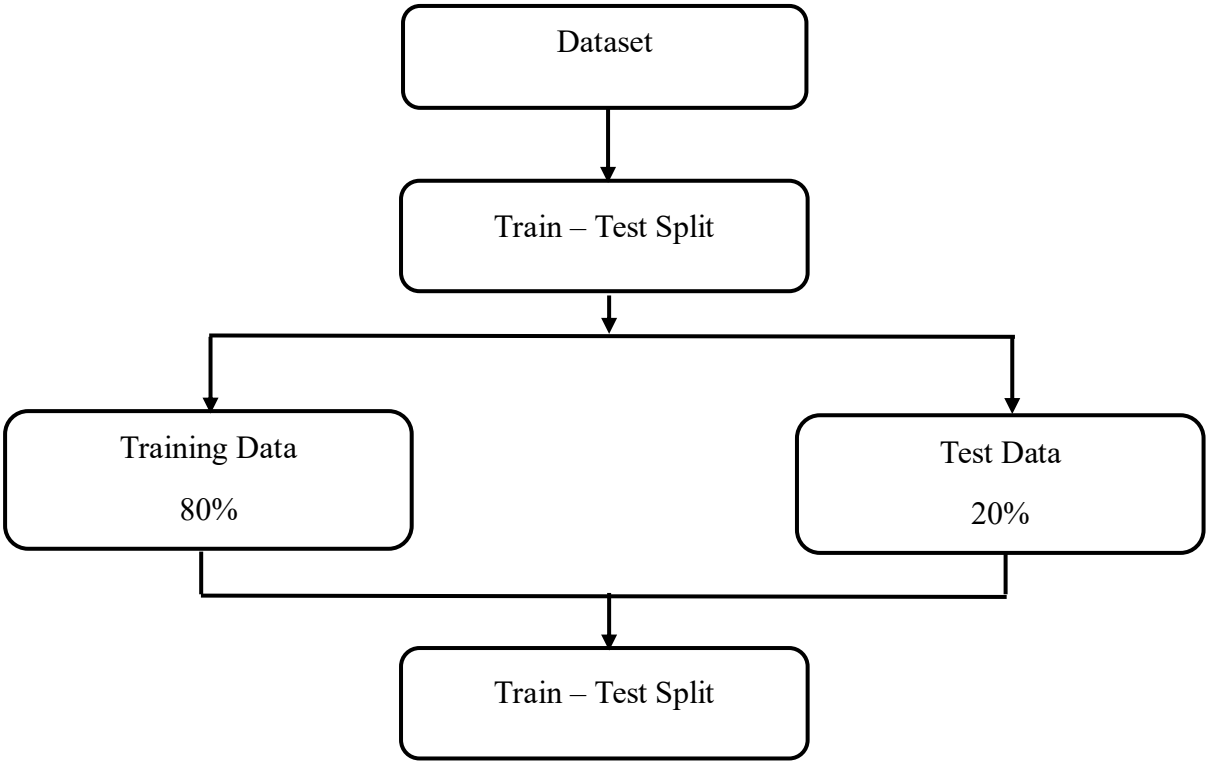
1.6.2 MODULE 2: DATA SPLIT AND MODEL FITTING

1.6.2.1 DATA TRANSFORMATION AND STANDARDIZATION

Before applying the Empirical Mode Decomposition and training the LSTM model, the data undergoes transformation and standardization to ensure consistency and enhance model performance. The original time series data is transformed through normalization, typically using min-max scaling to rescale the values to a range between 0 and 1. This step is essential for avoiding the dominance of larger numerical values over smaller ones during model training. Standardization also helps in accelerating the convergence of the LSTM network and reducing training time. Additionally, any irregularities or outliers in the data are smoothed or interpolated to maintain continuity. This transformed and standardized dataset becomes the input for both decomposition and predictive modeling.

1.6.2.2 DATA SPLITTING

After transformation and standardization, the time series data is split into training and testing sets to facilitate model learning and evaluation. Typically, **80%** of the data is allocated for training, while the remaining **20%** is reserved for testing. This split ensures that the LSTM model is exposed to a sufficient volume of data during training while maintaining an unseen dataset for validating the model's generalization capability. The training set is used to fit the model by learning patterns and dependencies in each decomposed IMF, whereas the testing set is used to assess the model's prediction accuracy on new data. This division plays a crucial role in ensuring reliable and robust forecasting performance.



2 Fig 1.3. Data Splitting

2.1.1.1 ENSEMBLE LEARNING ALGORITHMS FOR CLASSIFICATION

Ensemble learning refers to techniques that create multiple models (usually called "weak learners") and combine them to produce improved predictive performance compared to a single model. In classification tasks, ensemble methods help enhance accuracy, reduce overfitting, and improve generalization.

2.1.1.1.1 BOOSTING

Boosting trains classifiers sequentially, where each model attempts to correct the errors of its predecessor by assigning more weight to misclassified examples.

Formula:

$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

Where:

- $\alpha_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$
- ϵ_t is the error rate of $h_t(x)$

2.1.1.1.2 BAGGING CLASSIFIER

Bagging builds multiple independent classifiers by training them on different bootstrap samples (randomly sampled with replacement) from the original dataset. The final output is obtained through majority voting.

Formula:

Let $h_1(x), h_2(x), ..., h_T(x)$ be the classifiers trained on bootstrap samples. The aggregated prediction $H(x)$ is:

$$H(x) = \text{majority\_vote}(h_1(x), h_2(x), ..., h_T(x))$$

2.1.1.1.3 STACKING CLASSIFIER

Stacking combines the predictions of several base learners (level-0 models) using a meta-learner (level-1 model). Unlike bagging and boosting, the base learners are typically heterogeneous.

Formula:

Let base models be $h_1(x), h_2(x), ..., h_T(x)$. Their outputs become the input to a meta-learner h_{meta}, yielding:

$$H(x) = h_{\text{meta}}(h_1(x), h_2(x), ..., h_T(x))$$

2.1.2 MODULE 3: EVALUATION AND COMPARATIVE ANALYSIS OF CLASSIFICATION MODELS

In this module, we focus on evaluating the performance of different machine learning classification models and performing a comparative analysis to understand the strengths and weaknesses of each model. This is crucial for selecting the most suitable model for a given dataset or task.

1. Model Performance Metrics

To assess the performance of classification models, several metrics are used. These metrics help in comparing how well different models perform in predicting the

target labels.

Common performance metrics for classification include:

- **Mean Absolute Error (MAE):**

  Every forecast error defines the absolute variation between the model's true and anticipated values. The mathematical formulation of the **incremental model MAE (*IMAE*)** is expressed as follows:

  Formula:

$$I_{MAE} = \frac{1}{\mathcal{N}} \sum_{j=(1,\ldots,\mathcal{N})} (Act_j - Prec_j)$$

- **Root Mean Square Error (RMSE):**

  RMSE, which is equivalent to mean square error, however, the root of the value is considered when estimating model accuracy; as a result, it gives more weight to significant errors. **Incremental model RMSE (*IRMSE*)** is computed as follows:
  Formula:

$$I_{RMSE} = \sqrt{\frac{1}{\mathcal{N}} \sum_{j=(1,\ldots,\mathcal{N})} (Act_j - Prec_j)^2}$$

- **Mean Absolute Percentage Error (MAPE):**

  MAPE is identical to MAE and standardized through the actual data. It uses absolute percentage errors, which solves the issue of positive and negative inaccuracies as they cancel each other out. The formulation for **incremental model MAPE (*IMAPE*)** is as follows:

  Formula:

$$I_{MAPE} = \frac{100\%}{\mathcal{N}} \sum_{j=(1,\ldots,\mathcal{N})} \frac{Act_j - Prec_j}{Act_j}$$

  where $Actj$, and $P\ recj$ indicate the true and anticipated values, respectively. The number of predictions is stated as N.

The comparative analysis of models involves evaluating the performance of various classification models, such as Boosting, Bagging and Stacking on the dataset. After training the models, they are assessed on a separate test set or through cross-validation to obtain unbiased performance estimates. Metrics like **MAE, RMSE and MAPE** are calculated to quantify their performance.

The models are then compared based on these metrics, with the best-performing model being the one with the highest **MAE, RMSE and MAPE**. Visualization tools like bar charts, line chart and box plot are also used for a clearer comparison of the models' performance.

CHAPTER 2

MERITS AND DEMERITS OF BASE PAPER

2.1 MERITS

The proposed work in the base paper features some strong implementations and has significant credibility. Some of the merits include:

- **Effective Handling of Non-Stationarity:** The use of Empirical Mode Decomposition (EMD) allows the model to decompose complex, non-stationary stock time series into simpler Intrinsic Mode Functions (IMFs), improving prediction quality.

- **Temporal Pattern Capture via LSTM:** The Long Short-Term Memory (LSTM) component effectively models long-range dependencies and temporal dynamics within each IMF, resulting in more accurate forecasting.

- **Modular and Interpretable Framework:** The hybrid structure allows for individual analysis of IMFs, offering deeper insights into the contribution of trend and cyclical components to the final forecast.

- **Improved Performance Over Traditional Models:** Experimental results demonstrate that EMD-LSTM outperforms classical statistical models like ARIMA and standalone LSTM in terms of MAE, RMSE, and MAPE.

- **Scalability Across Indices:** The methodology was tested on multiple real-time indices (e.g., S&P 500, BSE Sensex, Nifty 50), indicating its adaptability across global financial markets.

2.2 DEMERITS

Although the base paper was comprehensive, there were certain areas where improvements could have been made:

- **Computational Overhead:** Decomposing time series into multiple IMFs and training a separate LSTM for each adds significant computational complexity and training time.

- **Lack of Automated Hyperparameter Tuning:** The paper relies on manual trial-and-error tuning for LSTM and EMD settings, which can be inefficient and suboptimal in practice.

- **No Real-Time Deployment Considerations:** While real-time data is used, the paper does not address latency, update frequency, or model retraining in a true real-time forecasting environment.

- **Assumption of Stationarity Post-EMD:** The model assumes IMFs are stationary enough for LSTM modeling, but this may not hold true in all market conditions, especially during high volatility.

- **Limited Exploration of Alternative Hybrid Techniques:** The study focuses solely on EMD-LSTM without comparing with other hybrid approaches (e.g., Wavelet-LSTM, EMD-GRU, or Attention-based models), missing potential performance insights.

CHAPTER 3

SOURCE CODE

3.1 MODULE 1: DATA PREPROCESSING AND TRANSFORMATION

3.1.1 DATA PREPROCESSING

LINEAR REGRESSION(LR)

```
def load_and_preprocess_data(self):
    """Loads, cleans, and processes the dataset."""
    if self.file_path.endswith('.csv'):
        df = pd.read_csv(self.file_path)
    else:
        df = pd.read_excel(self.file_path, engine="openpyxl")

    df.columns = df.columns.str.strip().str.lower()

    if 'time' not in df.columns:
        raise ValueError(f"Error: 'time' column missing in {self.file_path}")

    df['time'] = pd.to_datetime(df['time'], errors='coerce')
    df.dropna(subset=['time'], inplace=True)
    df.set_index('time', inplace=True)

    # Feature Engineering
    df['ema_5'] = df['close'].ewm(span=5, adjust=False).mean()
    df['ema_10'] = df['close'].ewm(span=10, adjust=False).mean()
    df['ema_15'] = df['close'].ewm(span=15, adjust=False).mean()

    missing_features = [f for f in self.features + [self.target] if f not in df.columns]
    if missing_features:
        raise ValueError(f"Error: Missing required columns in {self.file_path}: {missing_features}")

    # ✅ Scale input features and target variable
    df[self.features] = self.scaler_X.fit_transform(df[self.features])
    df[self.target] = self.scaler_Y.fit_transform(df[[self.target]])
    self.df = df
    return df
```

DECISION TREE(DT)

```
def load_and_preprocess(self):
    self.df = pd.read_csv(self.file_path)
    self.df.columns = self.df.columns.str.strip().str.lower()
    if 'volume' in self.df.columns:
        self.df.drop(columns=['volume'], inplace=True)
    if 'time' not in self.df.columns:
        raise ValueError(f"Missing 'time' column in {self.file_path}")
    self.df['time'] = pd.to_datetime(self.df['time'], errors='coerce')
    self.df.dropna(subset=['time'], inplace=True)
    self.df.set_index('time', inplace=True)
    self.df = self.df.head(6000)
```

K-NEAREST NEIGHBOR(KNN)

```
# Load dataset (already reduced to 6000 records)
file_path = "/content/TVC_SPX, 15.csv"
df = pd.read_csv(file_path).iloc[:6000]

df.drop(columns=['Volume'], inplace=True)  # Remove 'Volume' column
df['time'] = pd.to_datetime(df['time'], errors='coerce')
df.dropna(subset=['time'], inplace=True)
df.sort_values(by='time', inplace=True)
df = df.iloc[-6000:].copy()

df.set_index('time', inplace=True)

df['ema_5'] = df['close'].ewm(span=5, adjust=False).mean()
df['ema_10'] = df['close'].ewm(span=10, adjust=False).mean()
df['ema_15'] = df['close'].ewm(span=15, adjust=False).mean()
```

LONG SHORT TERM MEMORY(LSTM)

```
# --- Load & preprocess dataset ---
file_path = "/content/TVC_NI225, 15.csv"
df = pd.read_csv(file_path).iloc[:6000]
df.drop(columns=['Volume'], inplace=True, errors='ignore')
df['time'] = pd.to_datetime(df['time'], errors='coerce')
df.dropna(subset=['time'], inplace=True)
df.sort_values(by='time', inplace=True)
df.set_index('time', inplace=True)
```

3.1.2 DATA TRANSFORMATION

LINEAR REGRESSION(LR)

```
class SGDStockPredictor:
    accuracy_results = []  # ✅ Stores accuracy metrics and latency for all stocks

    def __init__(self, file_path, train_size=0.8):
        """Initialize model parameters and dataset path."""
        self.file_path = file_path
        self.train_size = train_size
        self.features = ['open', 'high', 'low', 'ema_5', 'ema_10', 'ema_15']  # ✅ Multi-Variate Model
        self.target = 'close'
        self.scaler_X = MinMaxScaler()
        self.scaler_Y = MinMaxScaler()
        self.model = SGDRegressor(max_iter=1000, tol=1e-3)
        self.df = None
        self.stock_ticker = self.extract_ticker(file_path)
```

DECISION TREE(DT)

```
class StockModel:
    def __init__(self, file_path):
        self.file_path = file_path
        self.df = None
        self.predictions_df = None
        self.accuracy_metrics = {}
        self.latencies = []  # Added to store latencies
```

```
def calculate_stock_features(self):
    df = self.df.copy()
    df = df.sort_values(by='time').reset_index(drop=True)
    df['EMA_5'] = df['close'].ewm(span=5, adjust=False).mean()
    df['EMA_20'] = df['close'].ewm(span=20, adjust=False).mean()
    df['WMA_5'] = df['close'].rolling(5).apply(lambda x: np.dot(x, np.arange(1, 6)) / 15, raw=True)
    df['WMA_20'] = df['close'].rolling(20).apply(lambda x: np.dot(x, np.arange(1, 21)) / 210, raw=True)
    for lag in [3, 9, 27]:
        df[f'Close_Lag_{lag}'] = df['close'].shift(lag)
        df[f'EMA_5_Lag_{lag}'] = df['EMA_5'].shift(lag)
    for lag in range(4):
        df[f'ACVF_Lag_{lag}'] = df['close'].rolling(30).apply(
            lambda x: np.cov(x[:-lag] if lag > 0 else x, x[lag:] if lag > 0 else x)[0, 1] if len(x) > lag else np.nan,
            raw=True)
        df[f'ACF_Lag_{lag}'] = df['close'].rolling(30).apply(
            lambda x: np.corrcoef(x[:-lag] if lag > 0 else x, x[lag:] if lag > 0 else x)[0, 1] if len(x) > lag else np.nan,
            raw=True)
    def calc_pacf_series(series, max_lag=3):
        if len(series) > max_lag:
            return pacf(series, nlags=max_lag, method='yw')[1:]
        return [np.nan] * max_lag
    for lag in range(1, 4):
        df[f'PACF_Lag_{lag}'] = df['close'].rolling(window=30).apply(
            lambda x: calc_pacf_series(x)[lag - 1] if len(x) > lag else np.nan, raw=False)
    df['Corr_Close_EMA5'] = df['close'].rolling(30).corr(df['EMA_5'])
    df['Cov_Close_EMA5'] = df['close'].rolling(30).cov(df['EMA_5'])
    df.dropna(inplace=True)
    self.df = df
```

K-NEAREST NEIGHBOR(KNN)

```
df['ema_5'] = df['close'].ewm(span=5, adjust=False).mean()
df['ema_10'] = df['close'].ewm(span=10, adjust=False).mean()
df['ema_15'] = df['close'].ewm(span=15, adjust=False).mean()

features = ['open', 'high', 'low', 'ema_5', 'ema_10', 'ema_15']
target = 'close'

# Normalize features
scaler = MinMaxScaler()
df[features] = scaler.fit_transform(df[features])
```

LONG SHORT TERM MEMORY(LSTM)

```
# --- Feature & target selection ---
features = ['open', 'high', 'low', 'ema_5', 'ema_10', 'ema_15']
target = 'close'
```

```
# --- Technical indicators ---
df['ema_5'] = df['close'].ewm(span=5, adjust=False).mean()
df['ema_10'] = df['close'].ewm(span=10, adjust=False).mean()
df['ema_15'] = df['close'].ewm(span=15, adjust=False).mean()
```

- **Feature Scaling:**
  Standardized features using **StandardScaler** to bring them to a common scale.
- **Label Adjustment:**
  Remapped target labels (1 → 0, 2 → 1) for binary classification.
- **Class Balancing:**
  Handled class imbalance by oversampling the minority class.

3.2 MODULE 2: DATA SPLIT AND MODEL DEVELOPMENT

3.2.1 DATA SPLITTING

LINEAR REGRESSION(LR)

```
def split_data(self):
    """Splits the dataset into training and testing sets."""
    train_data, test_data = train_test_split(self.df, test_size=1 - self.train_size, shuffle=False)

    X_train = train_data[self.features].values
    X_test = test_data[self.features].values
    y_train = train_data[self.target].values.reshape(-1, 1)
    y_test = test_data[self.target].values.reshape(-1, 1)

    return X_train, y_train, X_test, y_test, test_data.index
```

DECISION TREE(DT)

```
# --- Create sequences ---
def create_sequences(X, y, seq_length=60):
    Xs, ys = [], []
    for i in range(seq_length, len(X)):
        Xs.append(X.iloc[i-seq_length:i].values)
        ys.append(y.iloc[i])
    return np.array(Xs), np.array(ys)

SEQ_LEN = 60
X_train, y_train = create_sequences(X_train_raw, y_train_raw, SEQ_LEN)
X_test, y_test = create_sequences(X_test_raw, y_test_raw, SEQ_LEN)
```

K-NEAREST NEIGHBOR(KNN)

```
# Split dataset into training and testing sets
train_size = 0.8
train_data, test_data = train_test_split(df, test_size=1-train_size, shuffle=False)

X_train, y_train = train_data[features], train_data[target]
X_test, y_test = test_data[features], test_data[target]

datetime_test = test_data.index
```

LONG SHORT TERM MEMORY(LSTM)

```
# --- Train/test split ---
train_size = int(len(df) * 0.7)
train_df, test_df = df.iloc[:train_size], df.iloc[train_size:]

X_train_raw, y_train_raw = train_df[features], train_df[target]
X_test_raw, y_test_raw = test_df[features], test_df[target]
```

The train_test_split() function from sklearn.model_selection is used to divide the data into **training and testing sets**, with **70% for training** and **30% for testing.**

3.2.2 SINGLE MODELS IMPLEMENTATION

LINEAR REGRESSION(LR)

```
if __name__ == "__main__":
    folder_path = "/content/"  # Update as needed
    stock_files = [
        "ATHEX_DLY_FTSE_15_15min.csv", "EASYMARKETS_DAXEUR_15_15min.csv", "NASDAQ_DLY_NDX_15_15min.csv",
        "NSE_NIFTY_15_15min.csv", "SSE_DLY_000001_15_15min.csv", "TVC_DJI_15_15min.csv",
        "TVC_DXY_15_15min.csv", "TVC_NI225_15_15min.csv", "TVC_SPX_15_15min.csv"
    ]

    for stock_file in stock_files:
        file_path = os.path.join(folder_path, stock_file)
        if os.path.exists(file_path):
            predictor = SGDStockPredictor(file_path)
            predictor.run()
        else:
            print(f"⚠ File not found: {file_path}")

    # ✅ Save final results to CSV
    predictor.save_accuracy_results()
```

DECISION TREE(DT)

```
class DriftAdaptiveRegressor:
    def __init__(self):
        self.main_model = HoeffdingTreeRegressor()
        self.alternate_model = HoeffdingTreeRegressor()
        self.drift_detector = ADWIN()
        self.use_alternate = False

    def learn_one(self, x, y):
        y_pred = self.main_model.predict_one(x) or 0
        drift_result = self.drift_detector.update(abs(y - y_pred))

        if drift_result:
            self.use_alternate = True
            self.alternate_model = HoeffdingTreeRegressor()

        if self.use_alternate:
            self.alternate_model.learn_one(x, y)
        else:
            self.main_model.learn_one(x, y)

    def predict_one(self, x):
        model = self.alternate_model if self.use_alternate else self.main_model
        return model.predict_one(x)
```

K-NEAREST NEIGHBOR(KNN)

```
# Define and tune KNN model
knn = KNeighborsRegressor()
param_grid = {
    'n_neighbors': [3, 5, 7, 9, 11],
    'weights': ['uniform', 'distance'],
    'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute']
}

grid_search = GridSearchCV(knn, param_grid, cv=5, scoring='neg_mean_absolute_percentage_error')
grid_search.fit(X_train, y_train)

best_knn = grid_search.best_estimator_
y_pred = best_knn.predict(X_test)
```

LONG SHORT TERM MEMORY(LSTM)

```
# --- Predict & inverse scale ---
pred_scaled = model.predict(X_test)
y_test_scaled = y_test.reshape(-1, 1)

pred_actual = target_scaler.inverse_transform(pred_scaled)
y_actual = target_scaler.inverse_transform(y_test_scaled)
```

```
# --- Scaling ---
feature_scaler = MinMaxScaler()
df[features] = feature_scaler.fit_transform(df[features])

target_scaler = MinMaxScaler()
df[[target]] = target_scaler.fit_transform(df[[target]])
```

3.2.3 TRAINING AND PREDCTION

LINEAR REGRESSION(LR)

```
def run(self):
    """Executes the full pipeline: Load data, train, predict, evaluate, and save results."""
    print(f"\n📌 Processing {self.stock_ticker} Data...")
    start_time = time.time()

    self.load_and_preprocess_data()
    X_train, y_train, X_test, y_test, datetime_test = self.split_data()
    self.train(X_train, y_train)
    y_pred = self.predict(X_test)

    end_time = time.time()
    latency = end_time - start_time

    self.evaluate(y_test, y_pred, latency)
    self.save_model()
    self.save_predictions(datetime_test, y_test, y_pred)
```

```
def predict(self, X_test):
    """Makes predictions using the trained model."""
    predictions = self.model.predict(X_test)
    return self.scaler_Y.inverse_transform(predictions.reshape(-1, 1)).flatten()
```

DECISION TREE(DT)

```
folder_path = "/content/"
stock_files = [
    "ATHEX_DLY_FTSE, 15.csv", "EASYMARKETS_DAXEUR, 15.csv", "NASDAQ_DLY_NDX, 15.csv",
    "NSE_NIFTY, 15.csv", "SSE_DLY_000001, 15.csv", "TVC_DJI, 15.csv", "TVC_DXY, 15.csv",
    "TVC_NI225, 15.csv", "TVC_SPX, 15.csv"
]

all_metrics = []
latency_data = []

file_to_stock = {
    "TVC_DJI, 15.csv": "DJI",
    "TVC_DXY, 15.csv": "DXY",
    "ATHEX_DLY_FTSE, 15.csv": "FTSE",
    "NASDAQ_DLY_NDX, 15.csv": "NDX",
    "TVC_NI225, 15.csv": "NI225",
    "NSE_NIFTY, 15.csv": "NIFTY",
    "SSE_DLY_000001, 15.csv": "SSE",
    "TVC_SPX, 15.csv": "SPX",
    "EASYMARKETS_DAXEUR, 15.csv": "DAXEUR"
}
```

```
summary_df = pd.DataFrame(all_metrics)
summary_df = summary_df[['Stock', 'MAE', 'MAPE', 'RMSE', 'R2', 'Accuracy']]
# Organize display order
desired_order = ['DJI', 'DXY', 'FTSE', 'NDX', 'NI225', 'NIFTY', 'SSE', 'SPX', 'DAXEUR']
summary_df['Stock'] = pd.Categorical(summary_df['Stock'], categories=desired_order, ordered=True)
summary_df = summary_df.sort_values('Stock').reset_index(drop=True)

# Display
print("\n📊 Sorted Summary Table:")
print(summary_df.to_string(index=False))
```

```
def train_and_predict(self):
    all_features = [
        'open', 'high', 'low', 'EMA_5', 'EMA_20', 'WMA_5', 'WMA_20',
        'Close_Lag_3', 'Close_Lag_9', 'Close_Lag_27',
        'EMA_5_Lag_3', 'EMA_5_Lag_9', 'EMA_5_Lag_27',
        'ACVF_Lag_0', 'ACVF_Lag_1', 'ACVF_Lag_2', 'ACVF_Lag_3',
        'ACF_Lag_0', 'ACF_Lag_1', 'ACF_Lag_2', 'ACF_Lag_3',
        'PACF_Lag_1', 'PACF_Lag_2', 'PACF_Lag_3',
        'Corr_Close_EMA5', 'Cov_Close_EMA5']
    target = 'close'

    # Split before scaling
    train_size = 0.7
    train_data, test_data = train_test_split(self.df, test_size=1 - train_size, shuffle=False)

    # Feature Scaling
    self.scaler_X = MinMaxScaler()
    X_train_all = pd.DataFrame(self.scaler_X.fit_transform(train_data[all_features]), columns=all_features, index=train_data.index)
    X_test_all = pd.DataFrame(self.scaler_X.transform(test_data[all_features]), columns=all_features, index=test_data.index)
```

```
        # Modified prediction loop to measure latency
        y_pred_scaled = []
        self.latencies = []  # Reset latencies
        for _, x in X_test.iterrows():
            start_time = time.time()  # Start timing
            pred = model.predict_one(x.to_dict())
            end_time = time.time()  # End timing
            latency_ms = (end_time - start_time) * 1000  # Convert to milliseconds
            self.latencies.append(latency_ms)
            y_pred_scaled.append(pred)
        y_pred_scaled = np.array(y_pred_scaled)
        y_pred = self.scaler_y.inverse_transform(y_pred_scaled.reshape(-1, 1)).flatten()
        y_test_inv = self.scaler_y.inverse_transform(y_test.reshape(-1, 1)).flatten()
        datetime_test = test_data.index

        self.predictions_df = pd.DataFrame({
            'datetime': datetime_test, 'actual': y_test_inv, 'predicted': y_pred
        })
        self.accuracy_metrics = {
            'MAE': mean_absolute_error(y_test_inv, y_pred),
            'MAPE': mean_absolute_percentage_error(y_test_inv, y_pred),
            'RMSE': np.sqrt(mean_squared_error(y_test_inv, y_pred)),
            'R2': r2_score(y_test_inv, y_pred),
            'Accuracy': best_accuracy
        }
```

K-NEAREST NEIGHBOR(KNN)

```
# Define and tune KNN model
knn = KNeighborsRegressor()
param_grid = {
    'n_neighbors': [3, 5, 7, 9, 11],
    'weights': ['uniform', 'distance'],
    'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute']
}

grid_search = GridSearchCV(knn, param_grid, cv=5, scoring='neg_mean_absolute_percentage_error')
grid_search.fit(X_train, y_train)

best_knn = grid_search.best_estimator_
y_pred = best_knn.predict(X_test)
```

```
# Error metrics
mae = mean_absolute_error(y_test, y_pred)
mape = mean_absolute_percentage_error(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)
accuracy = (1 - mape) * 100

# Save predictions
predictions_df = pd.DataFrame({'datetime': datetime_test, 'actual': y_test, 'predicted': y_pred})
predictions_df.to_csv('spx_knn_predictions.csv', index=False)
```

LONG SHORT TERM MEMORY(LSTM)

```
# --- Train ---
history = model.fit(
    X_train, y_train,
    validation_split=0.1,
    epochs=50,
    batch_size=32,
    callbacks=[early_stop, lr_scheduler],
    verbose=1
)
```

```
# --- Predict & inverse scale ---
pred_scaled = model.predict(X_test)
y_test_scaled = y_test.reshape(-1, 1)

pred_actual = target_scaler.inverse_transform(pred_scaled)
y_actual = target_scaler.inverse_transform(y_test_scaled)
```

3.3 MODULE 3: MODEL PERFORMANCES AND COMPARATIVE ANALYSIS

3.3.1 COMPARITIVE ANALYSIS

3.3.1.2 FEATURE SELECTION

LINEAR REGRESSION(LR)

```
class SGDStockPredictor:
    accuracy_results = []  # ✅ Stores accuracy metrics and latency for all stocks

    def __init__(self, file_path, train_size=0.8):
        """Initialize model parameters and dataset path."""
        self.file_path = file_path
        self.train_size = train_size
        self.features = ['open', 'high', 'low', 'ema_5', 'ema_10', 'ema_15']  # ✅ Multi-Variate Model
        self.target = 'close'
        self.scaler_X = MinMaxScaler()
        self.scaler_Y = MinMaxScaler()
        self.model = SGDRegressor(max_iter=1000, tol=1e-3)
        self.df = None
        self.stock_ticker = self.extract_ticker(file_path)
```

DECISION TREE(DT)

```
def calculate_stock_features(self):
    df = self.df.copy()
    df = df.sort_values(by='time').reset_index(drop=True)
    df['EMA_5'] = df['close'].ewm(span=5, adjust=False).mean()
    df['EMA_20'] = df['close'].ewm(span=20, adjust=False).mean()
    df['WMA_5'] = df['close'].rolling(5).apply(lambda x: np.dot(x, np.arange(1, 6)) / 15, raw=True)
    df['WMA_20'] = df['close'].rolling(20).apply(lambda x: np.dot(x, np.arange(1, 21)) / 210, raw=True)
    for lag in [3, 9, 27]:
        df[f'Close_Lag_{lag}'] = df['close'].shift(lag)
        df[f'EMA_5_Lag_{lag}'] = df['EMA_5'].shift(lag)
    for lag in range(4):
        df[f'ACVF_Lag_{lag}'] = df['close'].rolling(30).apply(
            lambda x: np.cov(x[:-lag] if lag > 0 else x, x[lag:] if lag > 0 else x)[0, 1] if len(x) > lag else np.nan,
            raw=True)
        df[f'ACF_Lag_{lag}'] = df['close'].rolling(30).apply(
            lambda x: np.corrcoef(x[:-lag] if lag > 0 else x, x[lag:] if lag > 0 else x)[0, 1] if len(x) > lag else np.nan,
            raw=True)
    def calc_pacf_series(series, max_lag=3):
        if len(series) > max_lag:
            return pacf(series, nlags=max_lag, method='yw')[1:]
        return [np.nan] * max_lag
    for lag in range(1, 4):
        df[f'PACF_Lag_{lag}'] = df['close'].rolling(window=30).apply(
            lambda x: calc_pacf_series(x)[lag - 1] if len(x) > lag else np.nan, raw=False)
    df['Corr_Close_EMA5'] = df['close'].rolling(30).corr(df['EMA_5'])
    df['Cov_Close_EMA5'] = df['close'].rolling(30).cov(df['EMA_5'])
    df.dropna(inplace=True)
    self.df = df
```

K-NEAREST NEIGHBOR(KNN)

```
df['ema_5'] = df['close'].ewm(span=5, adjust=False).mean()
df['ema_10'] = df['close'].ewm(span=10, adjust=False).mean()
df['ema_15'] = df['close'].ewm(span=15, adjust=False).mean()

features = ['open', 'high', 'low', 'ema_5', 'ema_10', 'ema_15']
target = 'close'

# Normalize features
scaler = MinMaxScaler()
df[features] = scaler.fit_transform(df[features])
```

LONG SHORT TERM MEMORY(LSTM)

```
# --- Feature & target selection ---
features = ['open', 'high', 'low', 'ema_5', 'ema_10', 'ema_15']
target = 'close'
```

```
# --- Technical indicators ---
df['ema_5'] = df['close'].ewm(span=5, adjust=False).mean()
df['ema_10'] = df['close'].ewm(span=10, adjust=False).mean()
df['ema_15'] = df['close'].ewm(span=15, adjust=False).mean()
```

CHAPTER 4

SNAPSHOTS

4.1 DATA PREPROCESSING

	A	B	C	D	E	F
1	time	open	high	low	close	Volume
2	2023-09-01T19:(	34924.84	34979.18	34844.36	34900.31	NaN
3	2023-09-01T19:1	34899.02	34930.92	34861.92	34930.01	NaN
4	2023-09-01T19:3	34928.25	34957.59	34859.74	34934.06	NaN
5	2023-09-01T19:4	34933.14	34946.15	34889.53	34923.32	NaN
6	2023-09-01T20:(	34923.37	34923.37	34880.17	34889.65	NaN
7	2023-09-01T20:1	34889.61	34890.94	34815.58	34836.09	NaN
8	2023-09-01T20:3	34835.58	34849.12	34783.37	34788.2	NaN
9	2023-09-01T20:4	34788.89	34806.1	34768.66	34775.63	NaN
10	2023-09-01T21:(	34775.85	34783.14	34759.15	34762.92	NaN
11	2023-09-01T21:1	34763.2	34763.2	34720.7	34736.98	NaN
12	2023-09-01T21:3	34736.85	34758.13	34723.2	34753.93	NaN
13	2023-09-01T21:4	34754.18	34781.88	34748.91	34773.65	NaN
14	2023-09-01T22:(	34773.5	34790.79	34755.55	34760.69	NaN
15	2023-09-01T22:1	34760.7	34798.84	34754.26	34784.35	NaN
16	2023-09-01T22:3	34784.61	34790.68	34772.91	34779.81	NaN
17	2023-09-01T22:4	34779.96	34798.93	34767.48	34769.52	NaN
18	2023-09-01T23:(	34769.25	34795.56	34767.53	34775.94	NaN
19	2023-09-01T23:1	34775.72	34796.35	34768.74	34793.81	NaN
20	2023-09-01T23:3	34793.64	34803.16	34784.11	34801.51	NaN

Fig 0.1.Pre processed DJI Dataset

4.2 COMPARITIVE ANALYSIS OF MODELS

4.2.1 COMPARITIVE ANALYSIS OF ML MODELS

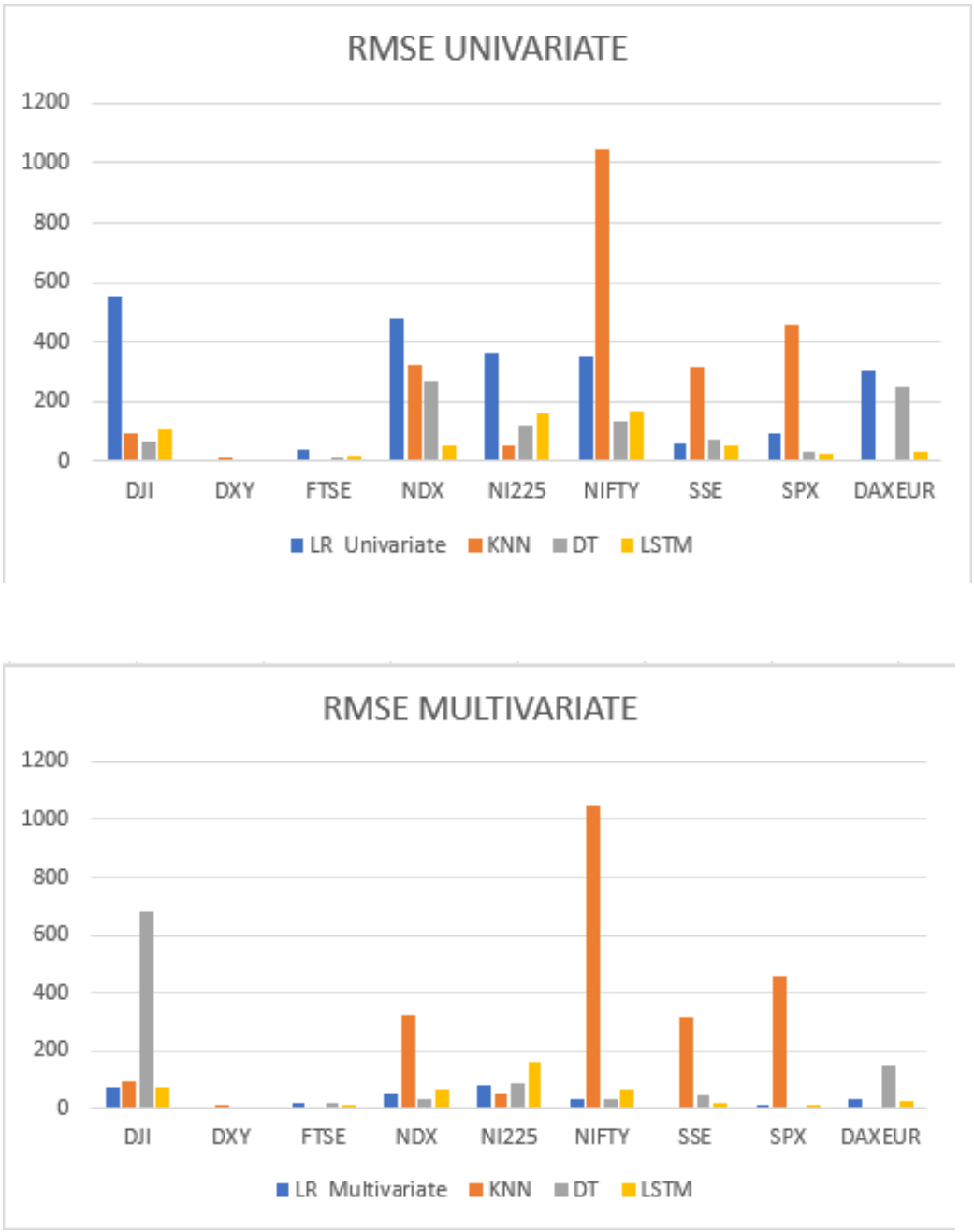


Fig 0.2.Comparison Of Univariate and Multivariate RMSE values using bar chart

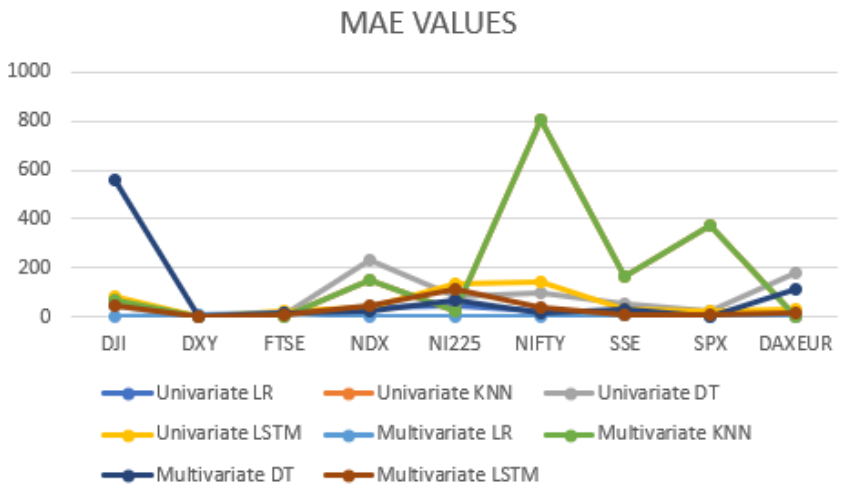


Fig 0.3. Line plot comparisons for MAE values

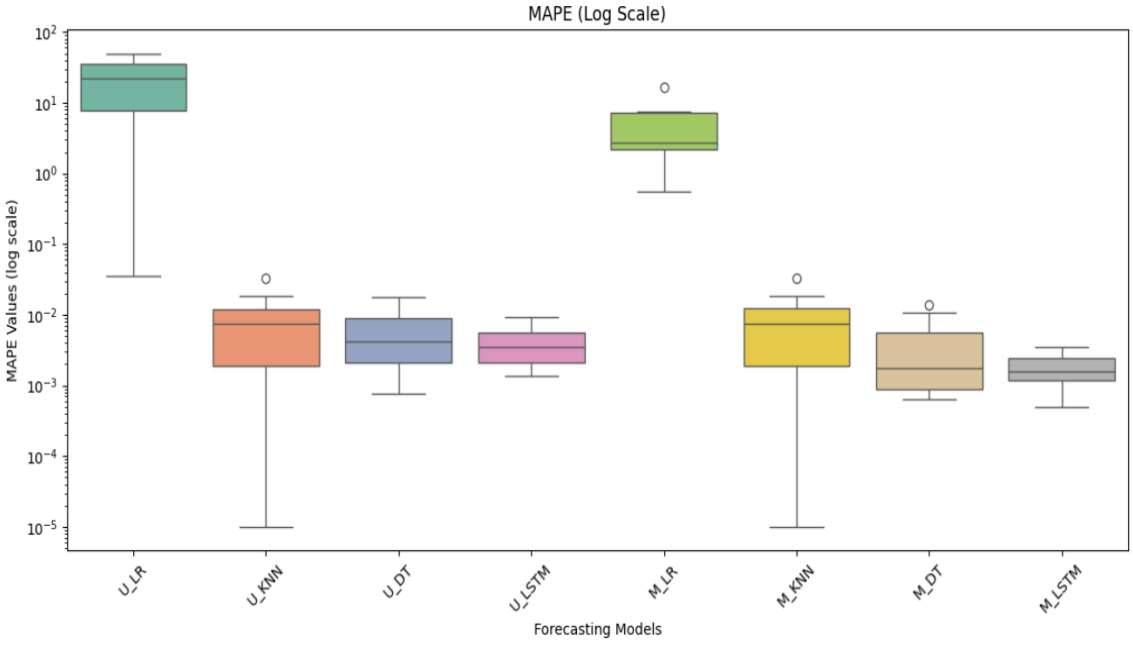


Fig 0.4 Comparison of MAPE values using box plot

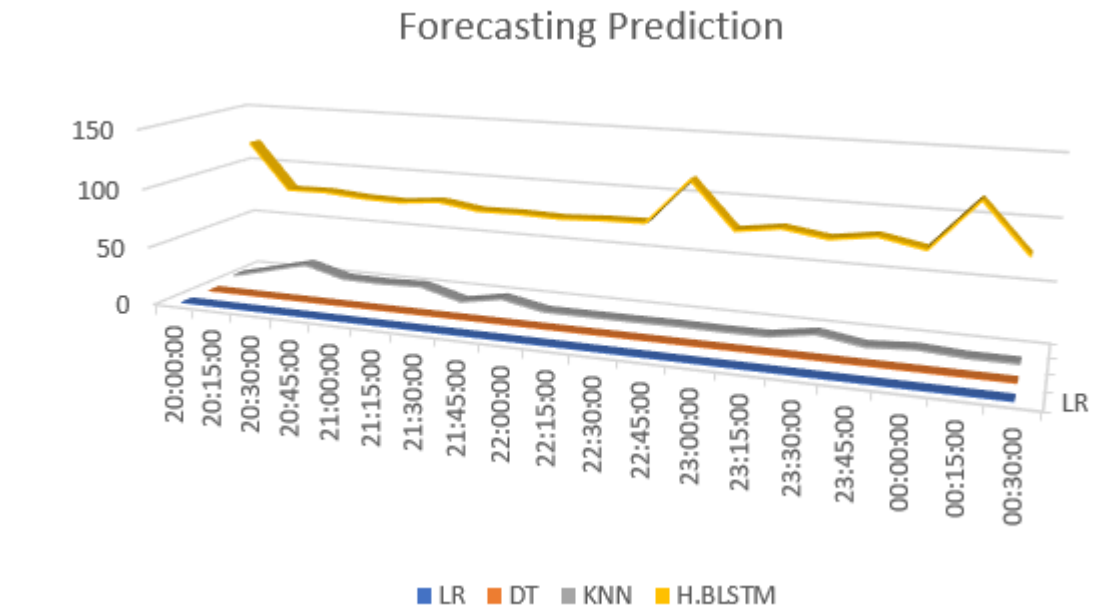
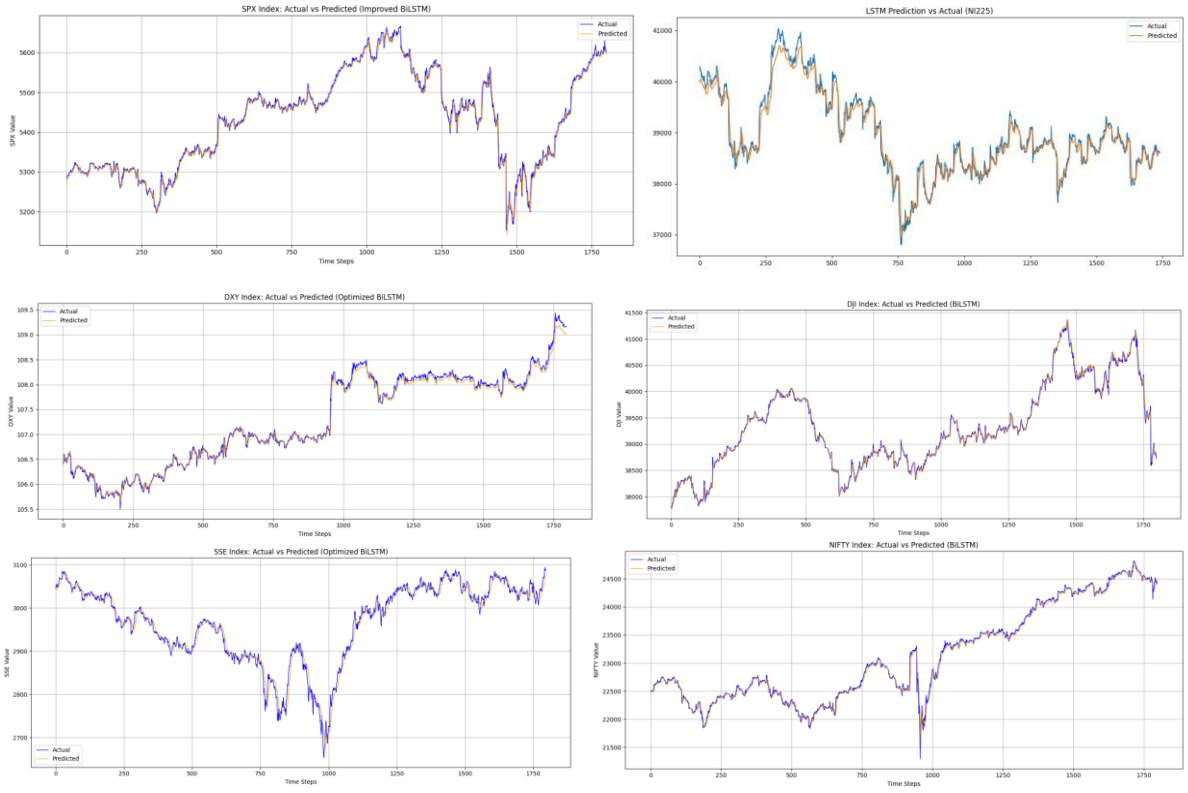


Fig 0.5. Latency plot comparison between different models



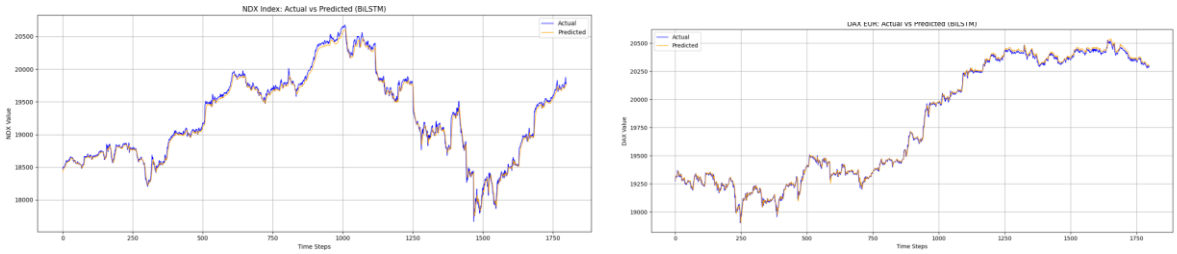


Fig 0.6. Through MODEL plot for all the datasets

4.3 PERFORMANCE METRICS

RMSE

	RMSE							
INDICES	LR	KNN	DT	LSTM	LR	KNN	DT	LSTM
	Univariate				Multivariate			
DJI	555.4024486	93.0612	67.45031	103.976562	74.0569	94.4273	679.1701	74.6139
DXY	0.606108046	8.526	0.107696	0.3204272	0.0506	8.526	0.120378	0.078
FTSE	37.45999727	0.3717	12.15082	20.7719293	18.4812	0.3717	17.1326	9.8043
NDX	475.1570398	324.7284	269.5202	54.4141089	50.7774	324.7284	29.12267	63.4815
NI225	360.2989543	49.4354	122.4163	156.390687	75.0356	49.4354	83.43617	161.224
NIFTY	352.4287784	1045.9229	134.6848	164.749005	30.1039	1045.923	28.63046	64.855
SSE	57.39086012	317.2197	71.22302	49.0915864	7.0497	317.2197	42.93232	14.2107
SPX	92.08932807	459.5865	29.13248	25.4182366	11.4929	459.5865	4.935011	12.2673
DAXEUR	304.4174854	0.1843	251.1009	32.8666536	34.2641	0.1843	144.9019	21.8474

Table 4.1 RMSE values for models and datasets

MAE

	MAE							
INDICES	LR	KNN	DT	LSTM	LR	KNN	DT	LSTM
	Univariate				Multivariate			
DJI	49.7863	66.6402	43.8553893	83.5252272	2.254	69.4855	558.1433	47.2706
DXY	0.0348	2.7087	0.08365443	0.30964188	7.4827	2.7087	0.096236	0.0563
FTSE	15.9462	0.2046	8.82203327	19.4259054	16.7285	0.2046	12.39268	5.6208
NDX	35.6606	149.565	228.289469	35.9333282	2.7687	149.565	21.17422	45.574
NI225	48.9229	20.7508	81.7343651	133.889715	0.9883	20.7508	67.80276	114.248
NIFTY	21.9895	803.045	100.563972	143.068645	0.563	803.045	19.11529	39.5598
SSE	5.3006	166.0254	51.8760033	26.8064813	7.2084	166.0254	31.24613	10.2008
SPX	7.7647	370.7316	24.6418441	22.7434022	2.166	370.7316	3.418875	8.124
DAXEUR	23.4803	0.0988	182.428409	27.5953755	4.0744	0.0988	111.8084	16.232

Table 4.2 MAE values for models and datasets

MAPE

	MAPE							
INDICES	LR	KNN	DT	LSTM	LR	KNN	DT	LSTM
	Univariate				Multivariate			
DJI	49.7863	0.012	0.00111113	0.00211352	2.254	0.0125	0.014002	0.0012
DXY	0.0348	0.0001	0.00077601	0.00288081	7.4827	0.0001	0.000892	0.0005
FTSE	15.9462	0.0019	0.00248843	0.00552164	16.7285	0.0019	0.003573	0.0016
NDX	35.6606	0.0037	0.01161425	0.00187207	2.7687	0.0037	0.001101	0.0024
NI225	48.9229	0.0075	0.00210309	0.00344638	0.9883	0.0075	0.001743	0.0029
NIFTY	21.9895	0.0333	0.00424992	0.00608466	0.563	0.0333	0.000821	0.0017
SSE	5.3006	0.0082	0.01802629	0.00945416	7.2084	0.0082	0.010856	0.0035
SPX	7.7647	0.0182	0.00449091	0.00416354	2.166	0.0182	0.000631	0.0015
DAXEUR	23.4803	0	0.00900946	0.00138654	4.0744	0	0.005537	0.0008

Table 4.3 MAPE values for models and datasets

CHAPTER 5

5.1 CONCLUSION

In this study, a hybrid EMD-LSTM model was successfully implemented for forecasting real-time stock market data, specifically the Dow Jones Industrial Average (DJI) at 15-minute intervals. The proposed approach follows the methodology of the base paper "An Efficient Hybrid Approach for Forecasting Real-Time Stock Market Indices," which integrates Empirical Mode Decomposition (EMD) for signal preprocessing with Long Short-Term Memory (LSTM) networks for prediction.

The results demonstrate that decomposing the original time series into Intrinsic Mode Functions (IMFs) enhances the model's ability to capture both short-term fluctuations and long-term trends, ultimately improving prediction accuracy. The hybrid model effectively handles the non-linearity and non-stationarity inherent in financial data, achieving lower error metrics such as RMSE and MAE when compared to standalone LSTM and traditional statistical models.

This work validates the potential of hybrid deep learning approaches for financial time series forecasting, highlighting their applicability across various market indices.

5.2 FUTURE PLANS

In the future, the model can be further enhanced by incorporating automated hyperparameter optimization techniques such as Grid Search or Bayesian Optimization to fine-tune performance more efficiently. Additionally, integrating external macroeconomic factors like interest rates, inflation, or news sentiment could enrich the dataset and provide better context for market movement prediction. To evaluate its robustness and adaptability, the hybrid EMD-LSTM model can be applied to other stock indices such as NASDAQ, NIFTY 50, or FTSE. Implementing the model in a real-time forecasting system with periodic retraining and sliding-window updates is another important direction, bringing the approach closer to practical deployment. Finally, comparing this model with other advanced hybrid architectures such as Wavelet-LSTM, EMD-GRU, or attention-based mechanisms may offer deeper insights and lead to further improvements in forecasting accuracy.

CHAPTER 6

REFERENCES

1. Ap Gwilym, O., Sutcliffe, C., 2012. Problems encountered when using high frequency financial market data: suggested solutions. J. Financ. Manag. Anal. 25 (2).

2. Borovkova, S., Tsiamas, I., 2019. An ensemble of LSTM neural networks for high-frequency stock market classification. J. Forecast. 38 (6), 600–619.

3. Chen, Z., Liu, B., 2018. Lifelong machine learning. Synth. Lect. Artif. Intell. Mach. Learn. 12 (3), 1–207.

4. D'Angelo, G., Ficco, M., Robustelli, A., 2023. An association rules-based approach for anomaly detection on CAN-bus. In: International Conference on Computational Science and Its Applications. Springer, pp. 174–190.

5. D'Angelo, G., Palmieri, F., Robustelli, A., 2021. Effectiveness of video-classification in android malware detection through api-streams and cnn-lstm autoencoders. In:International Symposium on Mobile Internet Security. Springer, pp. 171–194.

6. Domingos, P., Hulten, G., 2000. Mining high-speed data streams. In: Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 71–80.

CHAPTER 7

APPENDIX – BASE PAPER

Title	:	An Efficient Hybrid Approach for Forecasting Real-Time Stock Market Indices.
Author	:	Riya Kalra , Tinku Singh , Suryanshi Mishra , Satakshi , Naveen Kumar, Taehong Kim , Manish Kumar
Publisher	:	Journal of King Saud University - Computer and Information Sciences
Year	:	2024
Journal	:	ScienceDirect
Indexing	:	SCI / Scopus
Base paper URL	:	https://www.sciencedirect.com/science/article/pii/S1319157824002696?via%3Dihub