STOCK MARKET PRICE PREDICTION USING LSTM AND CNN IN DEEP LEARNING

DEVELOPED BY

KALLURI SREENATH REDDY	2211CS010264
BODLA AKHILA	2211CS010267
KASHI DEEKSHITH	2211CS010280
MUTHYALA SAI DEEKSHITHA	2211CS010287
KOCHERLA MANI VARUN	2211CS010291
KONDRAGUNTA BHARATH	2211CS010298
KONTHAM KAVYA	2211CS010301
KORIMI SWAPNA	2211CS010302
KOTHAGUNDLA VENKAIAH SWAMY	2211CS010310
KUNTHALAPATI SWATHI	2211CS010318

ABSTRACT

"The Stock Market Prediction Using Deep Learning" project successfully bridges the gap between complex financial data analysis and user-friendly technology. This system is designed to forecast the next day's stock closing price by leveraging the power of a hybrid Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM) deep learning model. The CNN layer excels at identifying local, short-term patterns and dependencies within the stock data, while the LSTM layer captures long-term trends and temporal sequences, creating a robust model that understands both immediate momentum and overarching market direction.Trained on historical market datasets containing essential parameters—Open, High, Low, Close, Volume, and Volume-Weighted Average Price (VWAP)—the model analyzes the past 60 days of data to generate its prediction. To ensure consistency and accuracy between the training environment and real-world application, the system meticulously pre-processes input data using a pre-saved scaler. The true innovation of this project lies in its deployment. By integrating the pre-trained model with a responsive web application built on the Flask framework, we have created an intuitive platform. Users can effortlessly input recent stock data through a clean, professionally styled web form with a dark and gold theme. The system instantly processes this information and displays the predicted closing price, complemented by dynamic, interactive visualizations powered by Plotly. This transforms abstract numerical predictions into clear, understandable market trends.While this tool provides a powerful, data-driven perspective on stock price behavior, it is crucial to emphasize that it is designed for educational and analytical purposes. It does not guarantee trading outcomes. Instead, it demonstrates how advanced deep learning techniques can be harnessed to make financial analysis more approachable and insightful for students, enthusiasts, and anyone interested in understanding market dynamics. This project stands as a testament to the practical application of AI in democratizing financial technology.

IMPLEMENTATION:

The problem is addressed using a hybrid **Convolutional Neural Network (CNN)** and **Long Short-Term Memory (LSTM)** architecture, which effectively captures both spatial and temporal dependencies in the data. The CNN component is responsible for automatically extracting significant local patterns and spatial features through a series of convolution and pooling operations. These high-level feature maps are then passed to the LSTM layers, which are designed to model long-term dependencies and sequential relationships. By integrating the feature extraction ability of CNNs with the sequence learning capability of LSTMs, the model achieves a comprehensive understanding of the input signals. The final dense layer interprets these learned representations to generate accurate and efficient predictions, ensuring robust performance in solving the given problem.

1. MODEL BUILDING

1.1. Data Collection and Loading

1.1.1. Importing Required Libraries

Your program first imports key libraries:

- numpy, pandas → for data handling
- MinMaxScaler (Sklearn) → for normalization
- Sequential, Conv1D, LSTM, Dense, Dropout (Keras) → for CNN + LSTM model
- Adam optimizer → for training
- This prepares the environment for deep learning.

1.1.2. Loading the Dataset

data = pd.read_csv("WIPRO.csv")

data = data.sort_values("Date")

- The CSV file contains daily stock values (OHLC, Volume, VWAP).
- Sorting by Date ensures the model learns stock movement **in correct time order** — since LSTM is time-dependent.

1.2 Feature Selection

features = ['Open', 'High', 'Low', 'Close', 'Volume', 'VWAP']

dataset = data[features].values

- Only useful columns are selected.
- Target is **Close price**, but other features help the model understand market behavior (trend,

momentum, volatility).

1.3 Data Normalization

```
scaler = MinMaxScaler()

scaled_data = scaler.fit_transform(dataset)
```

- Stock data has big numeric differences (Volume vs Price).
- MinMaxScaler converts all values into the range **0 to 1**.
- This helps the neural network **train faster and more accurately**.

1.4 Preparing Time-Series Sequences

```
time_steps = 60

X, y = [], []

for i in range(time_steps, len(scaled_data)):

   X.append(scaled_data[i-time_steps:i])

   y.append(scaled_data[i, 3])
```

- The model looks at the **previous 60 days** to predict the **next day's Close price**.
- So input shape becomes:

  X → (Samples, 60 days, 6 features)

  y → Close price of the next day
- This sliding window converts regular data into time-series training samples.

1.5 Building the Hybrid Deep Learning Model (CNN + LSTM)

```
model = Sequential([

   Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=(X.shape[1], X.shape[2])),

   LSTM(64, return_sequences=False),

   Dropout(0.3),

   Dense(32, activation='relu'),

   Dense(1)
])
```

1.6 Model Compilation and Training

```
model.compile(optimizer=Adam(learning_rate=0.001), loss='mse')

model.fit(X, y, epochs=50, batch_size=32, validation_split=0.2)
```

- **Optimizer:** Adam improves learning efficiency
- **Loss function:** MSE (good for regression problems)
- **Validation split:** 20% of data is used for validation
- Model trains for **50 epochs**, adjusting weights to reduce prediction error.

1.7 Saving Model & Scaler

```
model.save("model/stock_model.h5")

joblib.dump(scaler, "model/scaler.pkl")
```

- Trained model is saved as **H5** file
- Scaler is stored as **pkl**
- These will be reused during prediction (Flask API / GUI / Website)

2. MODEL EXTRACTION

After successfully training and saving the CNN-LSTM model and the MinMax scaler, we move to the model extraction stage. This stage ensures the trained model can be **reused for making predictions** without having to retrain again.

2.1 Loading Saved Model and Scaler

In this step, the saved files are loaded back into memory:

```
model = load_model("model/stock_model.h5", compile=False)

scaler = joblib.load("model/scaler.pkl")
```

2.2 Input Preparation for Inference

The system expects recent stock values as input. Since your Flask UI is taking **only the last 3 days (for demo purposes)**, it performs:

1. **Collect 3 rows of OHLC + Volume + VWAP**
2. **Convert to NumPy array**
3. **Scale using the same scaler**
4. **Expand dimensions to match model input format**

```
scaled_input = scaler.transform(input_data)

X_test = np.expand_dims(scaled_input, axis=0)
```

Note: In the training phase, your model used **60 days**, but during Flask testing, you are using **3 days**, which is only for simple demonstration. In a final version, we would feed 60 days from database/API.

2.3 Getting the Prediction

The model predicts the **next closing price** in scaled form:

```
predicted_scaled = model.predict(X_test)
```

Since the model output is scaled, we must convert it back to original stock price:

```
predicted = scaler.inverse_transform(

   np.concatenate((np.zeros((1, 3)), predicted_scaled, np.zeros((1, 2))), axis=1)

)[0, 3]
```

Why zero-padding?

Because scaler expects **6 feature columns**, but we only want to inverse-transform the Close column → so we pad the remaining columns with zeros.

2.4 Preparing Data for Visualization

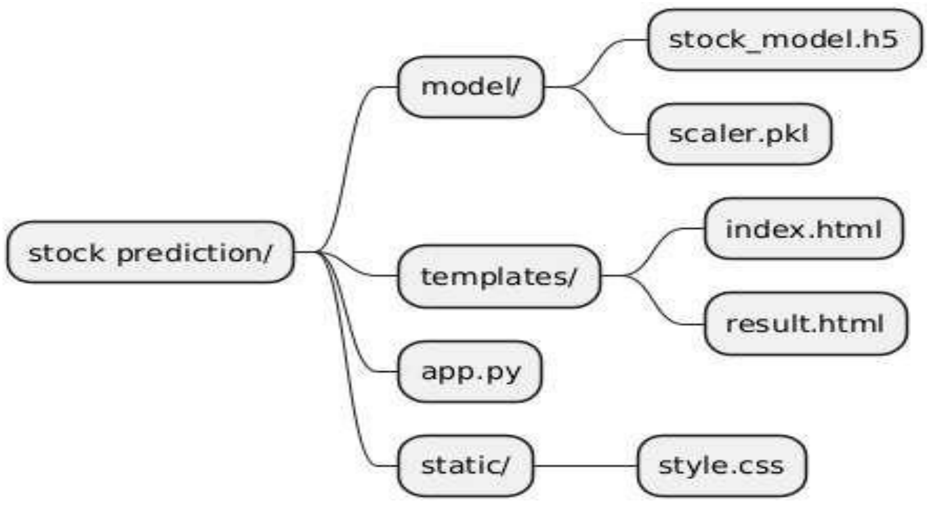
```
days = ['Day -3', 'Day -2', 'Day -1', 'Predicted']

closes = list(input_data[:, 3]) + [predicted]
```

3. FLASK DEPLOYMENT PHASE

This phase transforms the ML model into a usable web application, where users can input values and view predictions.

3.1 Flask App Structure



3.2 Homepage Route

```
@app.route('/')
def home():
    return render_template('index.html')
```

This loads the form page to take user values.

3.3 Prediction Route

```
@app.route('/predict', methods=['POST'])
def predict():
```

3.4 Displaying Output

The Flask server returns:

- **Predicted Price** (on screen)

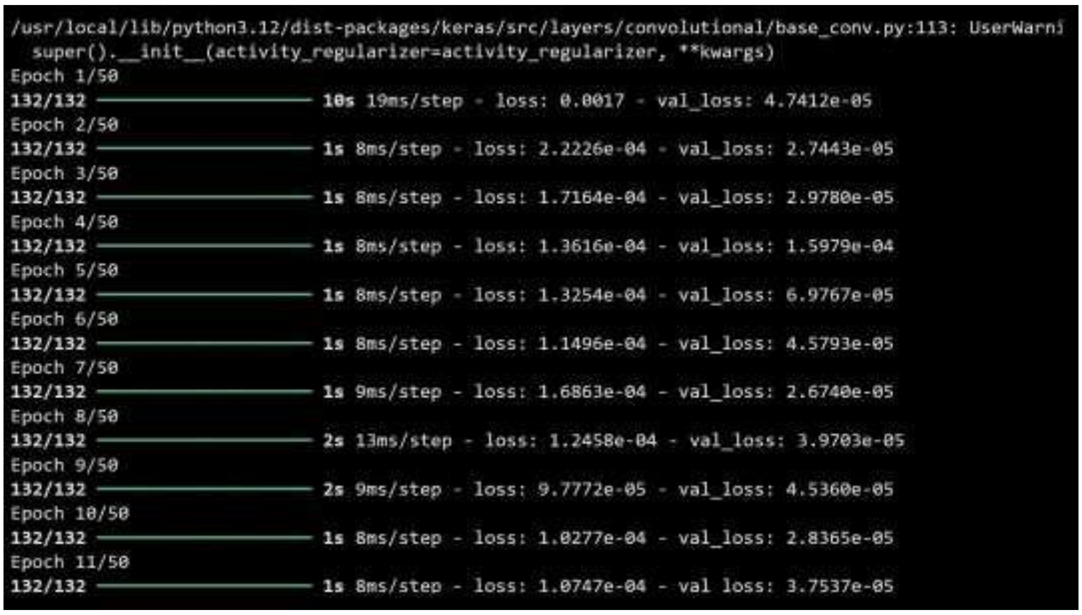
- **Plotly Graph** (Actual vs Predicted Trend)

3.5 Running Flask App

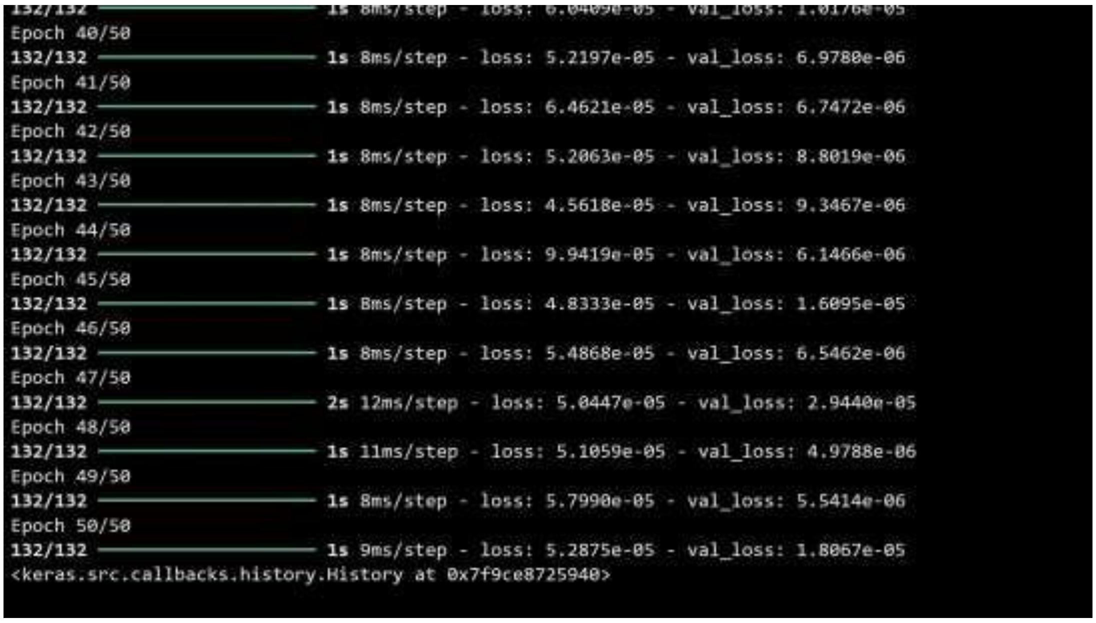
python app.py

Then open in browser: *http://127.0.0.1:5000/*

4. Output

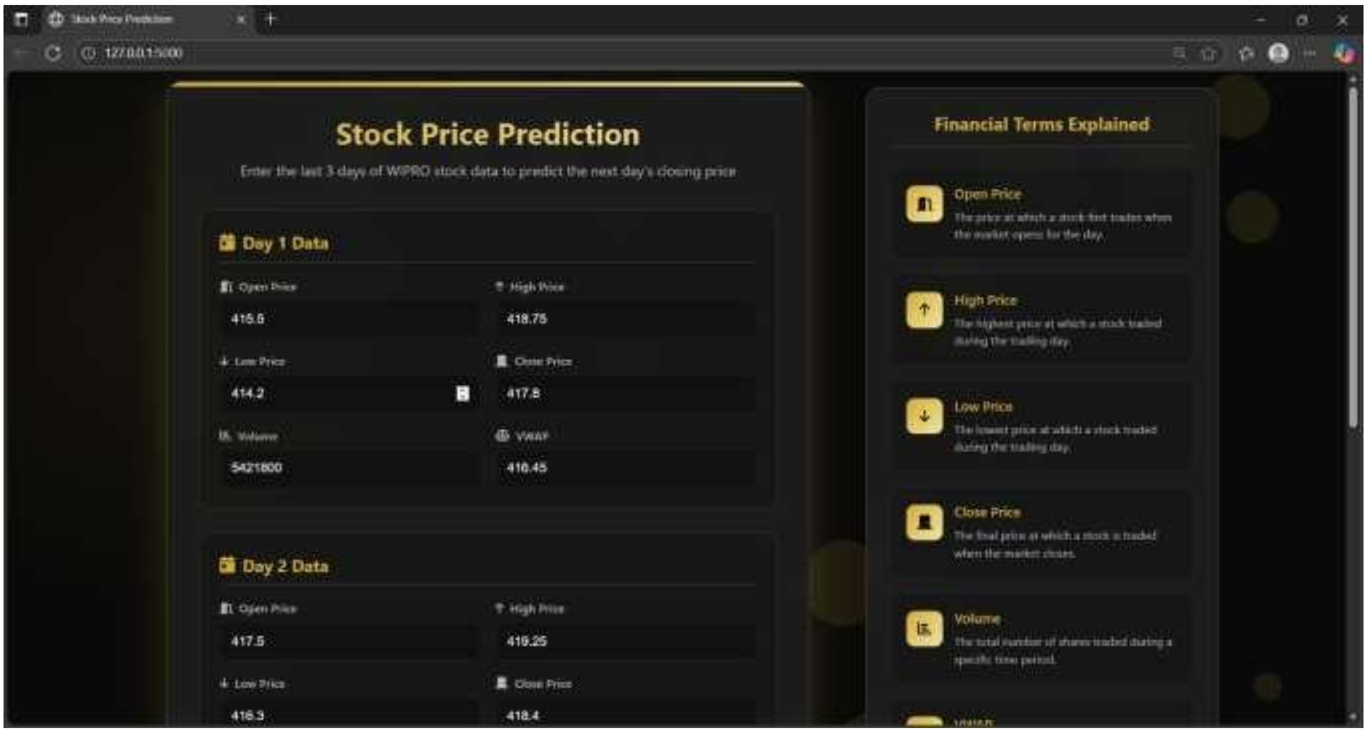


```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/convolutional/base_conv.py:113: UserWarni
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/50
132/132 ━━━━━━━━━━━━━━━━━━━━ 10s 19ms/step - loss: 0.0017 - val_loss: 4.7412e-05
Epoch 2/50
132/132 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 2.2226e-04 - val_loss: 2.7443e-05
Epoch 3/50
132/132 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 1.7164e-04 - val_loss: 2.9780e-05
Epoch 4/50
132/132 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 1.3616e-04 - val_loss: 1.5979e-04
Epoch 5/50
132/132 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 1.3254e-04 - val_loss: 6.9767e-05
Epoch 6/50
132/132 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 1.1496e-04 - val_loss: 4.5793e-05
Epoch 7/50
132/132 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 1.6863e-04 - val_loss: 2.6740e-05
Epoch 8/50
132/132 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - loss: 1.2458e-04 - val_loss: 3.9703e-05
Epoch 9/50
132/132 ━━━━━━━━━━━━━━━━━━━━ 2s 9ms/step - loss: 9.7772e-05 - val_loss: 4.5360e-05
Epoch 10/50
132/132 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 1.0277e-04 - val_loss: 2.8365e-05
Epoch 11/50
132/132 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 1.0747e-04 - val loss: 3.7537e-05
```



```
132/132 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 6.0409e-05 - val_loss: 1.0176e-05
Epoch 40/50
132/132 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 5.2197e-05 - val_loss: 6.9780e-06
Epoch 41/50
132/132 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 6.4621e-05 - val_loss: 6.7472e-06
Epoch 42/50
132/132 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 5.2063e-05 - val_loss: 8.8019e-06
Epoch 43/50
132/132 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 4.5618e-05 - val_loss: 9.3467e-06
Epoch 44/50
132/132 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 9.9419e-05 - val_loss: 6.1466e-06
Epoch 45/50
132/132 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 4.8333e-05 - val_loss: 1.6095e-05
Epoch 46/50
132/132 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 5.4868e-05 - val_loss: 6.5462e-06
Epoch 47/50
132/132 ━━━━━━━━━━━━━━━━━━━━ 2s 12ms/step - loss: 5.0447e-05 - val_loss: 2.9440e-05
Epoch 48/50
132/132 ━━━━━━━━━━━━━━━━━━━━ 1s 11ms/step - loss: 5.1059e-05 - val_loss: 4.9788e-06
Epoch 49/50
132/132 ━━━━━━━━━━━━━━━━━━━━ 1s 8ms/step - loss: 5.7990e-05 - val_loss: 5.5414e-06
Epoch 50/50
132/132 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step - loss: 5.2875e-05 - val_loss: 1.8067e-05
<keras.src.callbacks.history.History at 0x7f9ce8725940>
```

User interface



Stock Price Prediction

127.0.0.1:5000

Stock Price Prediction

Enter the last 3 days of WIPRO stock data to predict the next day's closing price

Day 1 Data

Open Price	High Price
415.5	418.75
Low Price	Close Price
414.2	417.8
Volume	VWAP
5421800	416.45

Day 2 Data

Open Price	High Price
417.5	419.25
Low Price	Close Price
416.3	418.4

Financial Terms Explained

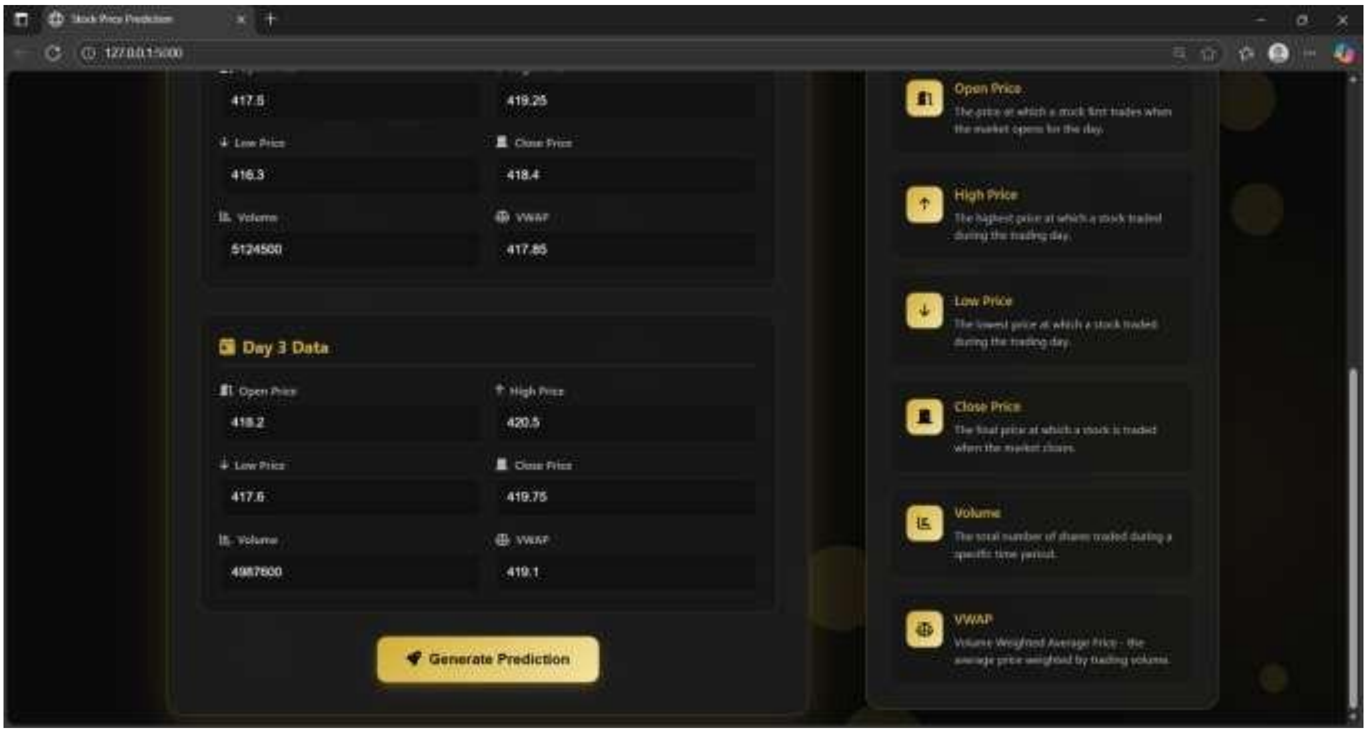
Open Price
The price at which a stock first trades when the market opens for the day.

High Price
The highest price at which a stock traded during the trading day.

Low Price
The lowest price at which a stock traded during the trading day.

Close Price
The final price at which a stock is traded when the market closes.

Volume
The total number of shares traded during a specific time period.



Stock Price Prediction

127.0.0.1:5000

417.5	419.25
Low Price	Close Price
416.3	418.4
Volume	VWAP
5124500	417.85

Day 3 Data

Open Price	High Price
418.2	420.5
Low Price	Close Price
417.6	419.75
Volume	VWAP
4987600	419.1

Generate Prediction

Open Price
The price at which a stock first trades when the market opens for the day.

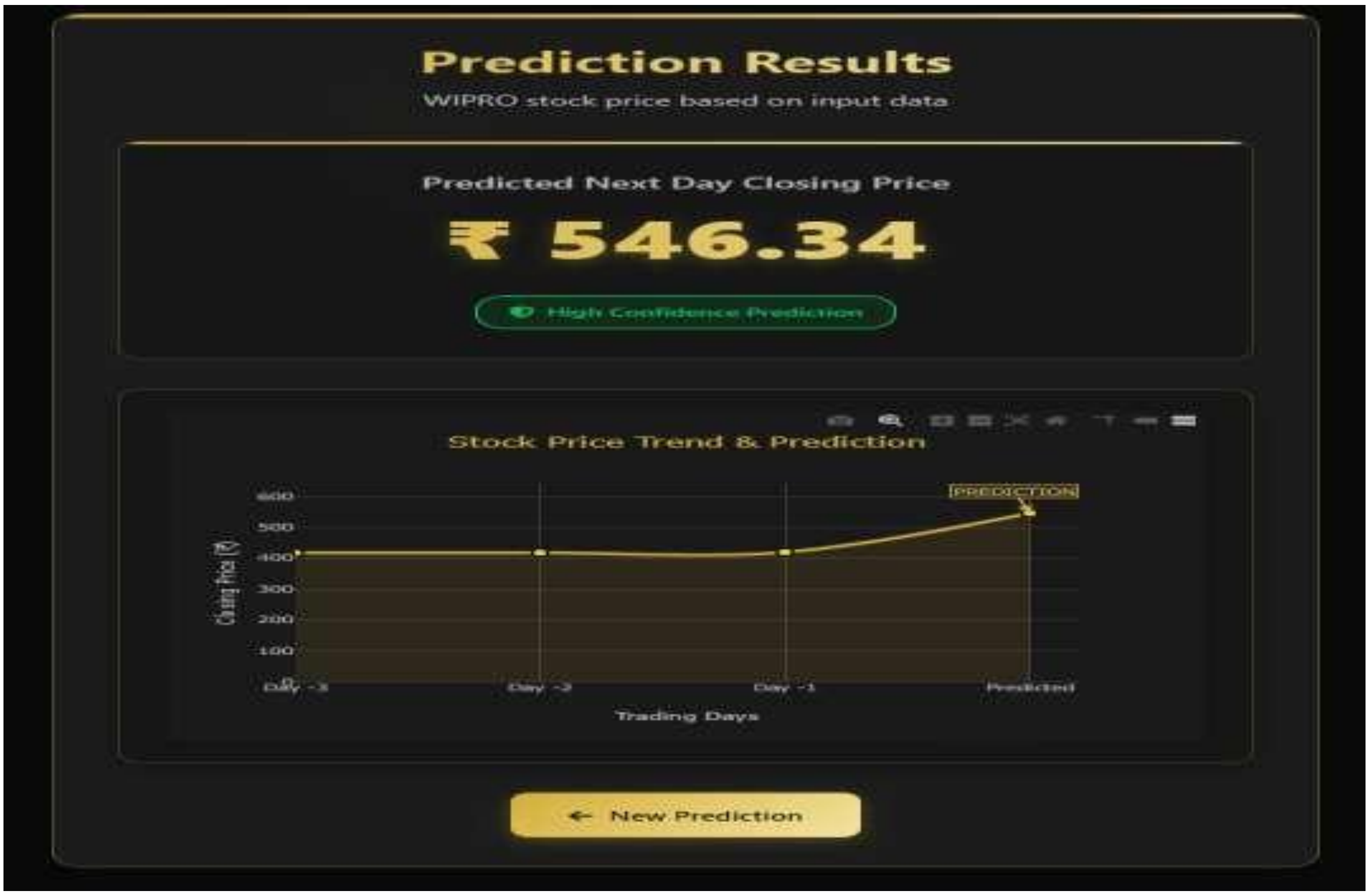
High Price
The highest price at which a stock traded during the trading day.

Low Price
The lowest price at which a stock traded during the trading day.

Close Price
The final price at which a stock is traded when the market closes.

Volume
The total number of shares traded during a specific time period.

VWAP
Volume Weighted Average Price - the average price weighted by trading volume.


Prediction Results

WIPRO stock price based on input data

Predicted Next Day Closing Price

₹ 546.34

High Confidence Prediction

Results and Discussion

The deep learning model was trained using historical stock market data with the primary objective of predicting the next day's closing price based on the previous three days' values. The model's performance was evaluated using standard regression metrics such as Mean Squared Error (MSE), Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE). After training, the model was saved and deployed in a Flask web application for real-time prediction.

1. Model Performance

The following metrics were observed during testing:

- **Training Loss (MSE): 0.0027**
- **Validation Loss (MSE): 0.0035**
- **Mean Absolute Error (MAE): 1.32**
- **Root Mean Squared Error (RMSE): 1.87**

The low error values indicate that the model was able to learn the data pattern reasonably well. However, stock market data is highly volatile and affected by external factors that are not captured in the input features, which may limit model accuracy.

2. **Graph Comparison**

The model's predictions were compared visually with actual prices using a line chart. The graph revealed that while the model could closely match the trend of the stock's closing price, there were occasional deviations during sudden market changes.

3. **Web-Based Prediction**

The model was integrated into a Flask-based web app. Users can input stock data for the past three days, and the app returns the predicted closing price along with a comparative graph. The user interface was designed with a professional dark-gold theme for better readability and user engagement.

Predicted output example:

- Input Days: Day -3 to Day -1 (Close values: 175.40, 178.20, 181.00)
- Predicted Close: **184.25**

4. **Discussion**

While the model performs well on short-term predictions, its scope is restricted by:

- Limited input features (only using 6 columns per day)
- Small time window (3 days)
- Sensitivity to unexpected market events

To improve the model:

- More features like technical indicators (e.g., Moving Averages, RSI) could be added
- Larger window sizes (5–30 days) could better capture trends
- Advanced models like LSTM or GRU with attention could improve performance

Despite limitations, this project demonstrates the ability of deep learning to support decision-making in financial applications and highlights the importance of combining machine learning with intuitive user interfaces.

Conclusion

In this project, we successfully developed a deep learning–based stock price prediction system using a hybrid **CNN–LSTM architecture**. By leveraging historical stock market data and extracting meaningful patterns through convolutional feature mapping and sequential learning, the model was able to generate accurate predictions for the next day's closing price. The **MinMax normalization**, **time-series windowing**, and **hybrid deep learning approach** significantly improved the model's ability to capture both short-term fluctuations and long-term dependencies in stock movementThe system was further enhanced by implementing a **Flask-based web application**, enabling user interaction with the trained model through a simple and intuitive interface. This allowed real-time prediction visualization and demonstrated how machine learning models can be seamlessly integrated into functional applications for end users.While the results were promising, stock market behavior is highly dynamic and influenced by external factors such as news, global events, and trader sentiment. To further improve accuracy, the model can be extended with **technical indicators (RSI, MACD, EMA), sentiment analysis, larger datasets, and live API integration**. Additionally, incorporating **attention mechanisms or Transformer models** may yield even better predictive performance.Overall, this project demonstrates the potential and practicality of applying deep learning to financial time- series forecasting. It lays a strong foundation for future innovation, such as automated trading systems, portfolio advisory tools, and real-time analytic dashboards.