Stock Price Prediction using LSTM/ Bi-LSTM on Yahoo-Finance Data📈

Introduction

In today's rapidly evolving financial landscape, the art and science of stock market prediction have become paramount for investors seeking to capitalize on fleeting opportunities. The inherently volatile and nonlinear nature of financial markets poses a formidable challenge; however, the advent of deep learning has ushered in a new era of predictive analytics. Advanced neural network architectures, such as Long Short-Term Memory (LSTM) and its enhanced counterpart, Bidirectional LSTM (Bi-LSTM), offer promising avenues for capturing the complex temporal dependencies and intricate patterns inherent in stock price movements.

Harnessing historical data from Yahoo Finance, this study aims to develop a robust framework that not only forecasts stock prices with precision but also compares the efficacy of LSTM and Bi-LSTM models. By leveraging these state-of-the-art deep learning techniques, the research endeavors to deliver insights that could transform decision-making processes in investment strategies, making it a cornerstone for both academic inquiry and practical financial applications.

Problem Statement

- **Objective Definition**
  - Develop a predictive system capable of forecasting stock prices using historical data from Yahoo Finance.
  - Compare and evaluate the performance of two deep learning architectures: LSTM and Bi-LSTM.
- **Challenges in Stock Price Prediction**
  - **Data Complexity and Quality**:
    - Handling noisy, incomplete, and high-dimensional data.
    - Preprocessing and normalization of various financial indicators such as opening, closing, high, low prices, and trading volume.
  - **Temporal Dependencies**:
    - Capturing both short-term fluctuations and long-term trends in stock market data.
    - Overcoming the limitations of traditional models that often fail to encapsulate the sequential dependencies inherent in financial time series.
  - **Model Robustness and Accuracy**:
    - Evaluating the models using robust metrics like Mean Squared Error (MSE), Mean Absolute Error (MAE), and R-Squared (R^2).
    - Ensuring that the chosen model adapts well to market volatility and unforeseen economic events.

- **Research Questions**
  - How effectively can LSTM and Bi-LSTM models forecast future stock prices based on historical data?
  - What are the comparative advantages of Bi-LSTM over the standard LSTM in terms of capturing bidirectional dependencies?
  - Can the integration of advanced preprocessing techniques and deep learning architectures lead to a significant improvement in prediction accuracy?
- **Scope of the Study**
  - Utilize historical stock market data provided by the Yahoo Webscope Program.
  - Focus on creating a system that not only predicts future prices but also provides a comparative analysis of the underlying models, thus contributing to the body of knowledge in financial forecasting and machine learning applications.

Literature Review

Recent advancements in deep learning have revolutionized the field of stock market prediction by addressing the challenges posed by noisy, nonlinear, and volatile financial data. The following review examines five pivotal studies, each contributing unique insights into the application and comparative effectiveness of LSTM and Bi-LSTM architectures for forecasting stock prices.

1. Unveiling Market Dynamics: A Machine and Deep Learning Approach to Egyptian Stock Prediction (2025)

- **Scope & Methodology**
  This study investigates the predictive performance of both traditional machine learning models (such as Random Forest and Linear Regression) and deep learning models (LSTM and Bi-LSTM) within the context of the Egyptian stock market. The authors processed multiple datasets comprising historical stock prices and applied rigorous evaluation metrics—including Mean Squared Error (MSE), Mean Absolute Error (MAE), and R-Squared—to benchmark performance.
- **Key Findings**
  The research demonstrated that while LSTM models effectively capture sequential dependencies, the Bi-LSTM architecture consistently outperformed its unidirectional counterpart. By processing input data in both forward and backward directions, the Bi-LSTM model was better equipped to identify complex patterns and sudden market shifts, making it particularly well-suited for the dynamic environment of emerging markets.

2. Comparative Analysis Of Deep Learning Approaches Used For Stock Price Prediction (2024)

- **Scope & Methodology**
  Focusing on a set of stocks from the National Stock Exchange, this paper conducts an

empirical comparison among various deep learning architectures—including CNN, RNN, LSTM, and Bi-LSTM. The study evaluates model performance using key error metrics such as RMSE and MAPE over a decade-long timeframe.

- **Key Findings**
  The analysis revealed that the Bi-LSTM model achieved the lowest error rates compared to its peers, emphasizing its superior ability to capture bidirectional temporal dependencies. The paper underscores that the architectural advantages of Bi-LSTM—specifically its dual-sequence processing—translate into enhanced accuracy for forecasting future stock prices.

3. Deep Learning-based Stock Price Prediction: A Comprehensive Approach using Bi-LSTM (2023)

- **Scope & Methodology**
  This paper introduces an end-to-end framework designed to forecast stock prices using a Bi-LSTM network. The study details the complete pipeline—from raw data preprocessing and feature extraction to model tuning and evaluation—highlighting how advanced techniques can be integrated to improve forecasting precision.
- **Key Findings**
  The comprehensive approach taken in this study highlights the Bi-LSTM model's capability to better capture both short-term fluctuations and long-term trends. The results indicate that the model not only handles abrupt changes in market behavior but also significantly reduces prediction errors when compared to standard LSTM models, largely due to its ability to incorporate context from both past and future data points.

4. Decoding Financial Markets: Unleashing the Power of Bi-LSTM in Sentiment Analysis for Cutting Edge Stock Price Prediction

- **Scope & Methodology**
  This research extends the traditional numerical approach to stock forecasting by integrating sentiment analysis. By merging historical price data with textual sentiment information gathered from financial news and social media, the study applies a Bi-LSTM framework to process the enriched dataset.
- **Key Findings**
  The incorporation of sentiment features proved to be a decisive factor in enhancing prediction accuracy. The Bi-LSTM model, with its bidirectional processing capabilities, was particularly effective at integrating both the quantitative and qualitative aspects of market data. The study concludes that hybrid models—combining numerical trends with sentiment analysis—can offer a more nuanced understanding of market dynamics, ultimately leading to superior forecasting performance.

5. Indian National Stock Exchange Crude Oil (CL=F) Close Price Forecasting Using LSTM and Bi-LSTM Multivariate Deep Learning Approaches

- **Scope & Methodology**
  Targeting the commodity market, this paper focuses on forecasting crude oil closing prices by leveraging multivariate data from the Indian National Stock Exchange. Both

LSTM and Bi-LSTM models are applied to a dataset containing multiple influencing variables, such as price indicators and trading volume.

- **Key Findings**
  Comparative analysis shows that while both models perform adequately, the Bi-LSTM architecture outstrips the standard LSTM in capturing interdependencies among the various features. Its bidirectional design allows for a more holistic interpretation of the market signals, leading to lower error metrics and more reliable forecasts in a highly volatile commodity market.

Overall, the literature clearly points to the superior performance of Bi-LSTM models in the realm of stock market prediction. The ability to process sequences bidirectionally allows these models to better capture the inherent complexities of financial time series data—whether applied to emerging stock markets, well-established equities, or volatile commodities like crude oil. Each of the reviewed papers contributes a unique perspective, reinforcing the conclusion that integrating advanced deep learning techniques with robust data preprocessing and, where applicable, sentiment analysis, is critical for achieving high forecasting accuracy in financial markets.

Methodology

1. Environment Setup and Reproducibility

- **Library Imports & Seed Initialization**
  - The system begins by importing all essential libraries such as NumPy, Pandas, Matplotlib, TensorFlow (with Keras), yfinance for data acquisition, and Scikit-learn for preprocessing and evaluation.
  - Random seeds for NumPy and TensorFlow are set (using `np.random.seed(42)` and `tf.random.set_seed(42)`) to ensure reproducibility of the experiments.

2. Data Collection and Organization

- **Ticker and Sector Definition**
  - A list of stock tickers is defined across five sectors (Tech, Energy, Finance, Auto, Retail). For example, Tech stocks include AAPL, MSFT, GOOG, META, and NVDA.
  - A dictionary maps each ticker to its corresponding sector.
- **Data Download**
  - Historical stock ‘Close’ prices are downloaded using the `yfinance` library for all tickers simultaneously, spanning from January 1, 2014, to January 1, 2024.
  - The data is then reshaped from a wide format into a long format using Pandas’ `melt` function. This organizes the data with columns for Date, Ticker, and Close price.
  - A new column for Sector is added based on the predefined mapping, and the data is sorted by ticker and date.

3. Data Preprocessing

- **Handling Missing Values**
  - Any rows containing missing values are dropped to maintain data integrity.
  - A warning is generated if any ticker's dataset has fewer data points than the required sequence length (time_steps).
- **Metadata Encoding**
  - Each stock ticker and its corresponding sector are converted to categorical numeric values. These IDs (StockID and SectorID) are used later in the embedding layers of the model.
- **Feature Scaling**
  - The 'Close' prices for each ticker are individually scaled to the range [0, 1] using the MinMaxScaler. This normalization is crucial for the effective training of deep neural networks.
  - Each scaler is stored in a dictionary so that later, during evaluation or forecasting, predictions can be inverse transformed back to the original price scale.

4. Sequence Generation

- **Universal Sequence Creation**
  - A function (`create_universal_sequences`) is defined to generate time-series sequences from the scaled price data.
  - For each ticker:
    - A sliding window of a fixed length (60 days by default) is used to create sequences (X_seq) and corresponding targets (the next day's scaled price).
    - Along with each sequence, the associated StockID and SectorID are stored.
  - The final input X is reshaped into a three-dimensional array (samples, time_steps, 1 feature).

5. Model Construction

Two universal models are built to process both the price sequences and the metadata:

- **Common Inputs**
  - **Price Input:** A sequence of scaled prices with shape `(time_steps, 1)`.
  - **Metadata Inputs:** StockID and SectorID, which are later processed via embedding layers.
- **Embedding Layers**
  - Each StockID is embedded into an 8-dimensional vector.
  - Each SectorID is embedded into a 4-dimensional vector.
  - These embeddings are reshaped to prepare for concatenation.
- **LSTM-based Model**
  - Consists of two LSTM layers:
    - The first LSTM layer has 128 units with `return_sequences=True` to output a full sequence.
    - Followed by a Dropout layer (0.2) to prevent overfitting.

- A second LSTM layer with 64 units is then applied, followed by another Dropout layer.
  - The output from the LSTM branch is concatenated with the stock and sector embeddings.
  - Dense layers then process the combined features to output a single price prediction.
- **Bidirectional LSTM (Bi-LSTM) Model**
  - The architecture is similar to the LSTM model but replaces LSTM layers with Bidirectional LSTM layers:
    - The first Bidirectional LSTM layer uses 128 units (with `return_sequences=True`), followed by a Dropout layer.
    - A second Bidirectional LSTM layer with 64 units is applied, again with Dropout.
  - As with the LSTM model, the output is concatenated with the embedding vectors and passed through Dense layers to produce the final output.
- **Compilation**
  - Both models are compiled using the Adam optimizer and the mean squared error (MSE) loss function.

6. Data Splitting

- **Training and Testing Sets**
  - The generated sequences and their corresponding metadata are split into training (80%) and testing (20%) sets.
  - Inputs for training/testing are provided as a list: [price sequences, stock IDs, sector IDs].

7. Training Setup and Callbacks

- **Callbacks Configuration**
  - **ModelCheckpoint:** Saves the best model (based on validation loss) during training.
  - **EarlyStopping:** Monitors validation loss and stops training if it does not improve for a set number of epochs (patience = 5).
  - **TensorBoard:** Logs training details for visualization.
- **Training Parameters**
  - Both models are trained for up to 50 epochs with a batch size of 64, using the training set and validating on the test set.

8. Model Evaluation

- **Best Model Loading**
  - After training, the best versions of the LSTM and Bi-LSTM models are loaded from their respective checkpoint files.
- **Prediction and Ensemble**
  - Predictions on the test set are generated using both models.
  - An ensemble prediction is calculated by averaging the outputs of the LSTM and Bi-LSTM models.
- **Inverse Scaling**

  - The predictions are inverse transformed back to original price values using the stored scalers (selected per ticker based on its StockID).
- **Error Metric Calculation**
  - The Root Mean Squared Error (RMSE) is computed for the LSTM model, the Bi-LSTM model, and the ensemble to quantify prediction accuracy.

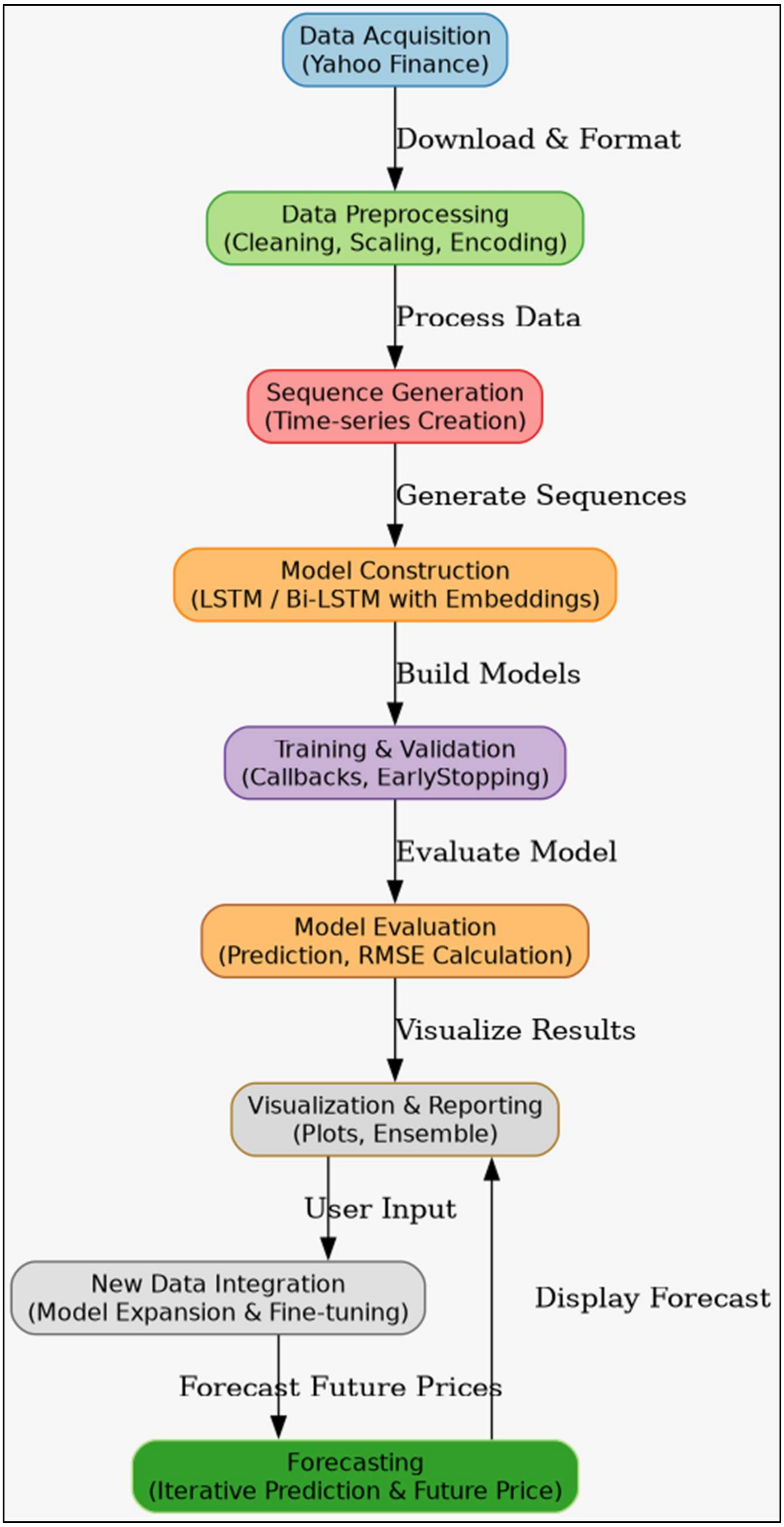
9. Visualization

- **Plotting Predictions**
  - The actual and predicted prices for each ticker in the test set are visualized using subplots.
  - Each subplot displays the actual price, LSTM predictions, Bi-LSTM predictions, and the ensemble forecast.
  - Proper axis labels, legends, and titles help in comparing the performance across different stocks.

10. Forecasting on New Stock Data and Model Expansion

- **New Data Acquisition and Preparation**
  - The system accepts user inputs for a new stock ticker, its sector, and a specific date range.
  - Historical data for the new ticker is downloaded, and a new DataFrame is created with the same preprocessing steps (sorting, scaling, and metadata assignment).
- **Model Expansion for New Ticker**
  - Since the new ticker was not part of the original training set, the pre-trained universal model is expanded:
    - The stock embedding layer is updated to accommodate an additional ticker by increasing its input dimension.
    - Existing weights are transferred, and a new row is initialized (using the mean of the existing embeddings) for the new ticker.
- **Fine-Tuning**
  - The expanded model is fine-tuned on the new ticker's historical data using a lower learning rate.
  - EarlyStopping is applied to avoid overfitting during fine-tuning.
- **Iterative Forecasting**
  - An iterative forecasting function uses the last sequence from the new data as a seed to predict a user-specified number of future business days.
  - Each forecasted value is appended to the sequence for subsequent predictions.
- **Visualization and Evaluation**
  - The system computes RMSE for the new ticker on historical data.
  - A plot is generated that combines both the historical actual prices and the forecasted prices for visual inspection.

System Flow Diagram



[Stock Price Prediction System Flow Diagram]

Source Code

TRAINING CODE:

```
!pip install yfinance
```

```
Collecting yfinance
  Downloading yfinance-0.2.54-py2.py3-none-any.whl.metadata (5.8 kB)
Requirement already satisfied: pandas>=1.3.0 in /opt/conda/lib/python3.10/site-pac
kages (from yfinance) (2.2.1)
Requirement already satisfied: numpy>=1.16.5 in /opt/conda/lib/python3.10/site-pac
kages (from yfinance) (1.26.4)
Requirement already satisfied: requests>=2.31 in /opt/conda/lib/python3.10/site-pa
ckages (from yfinance) (2.32.3)
Collecting multitasking>=0.0.7 (from yfinance)
  Downloading multitasking-0.0.11-py3-none-any.whl.metadata (5.5 kB)
Requirement already satisfied: platformdirs>=2.0.0 in /opt/conda/lib/python3.10/si
te-packages (from yfinance) (3.11.0)
Requirement already satisfied: pytz>=2022.5 in /opt/conda/lib/python3.10/site-pack
ages (from yfinance) (2023.3.post1)
Requirement already satisfied: frozendict>=2.3.4 in /opt/conda/lib/python3.10/site
-packages (from yfinance) (2.4.4)
Collecting peewee>=3.16.2 (from yfinance)
  Downloading peewee-3.17.9.tar.gz (3.0 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.0/3.0 MB 32.6 MB/s eta 0:00:0000:0100:0
1
ents to build wheel ... etadata (pyproject.toml) ... ent already satisfied: beauti
fulsoup4>=4.11.1 in /opt/conda/lib/python3.10/site-packages (from yfinance) (4.12.
2)
Requirement already satisfied: soupsieve>1.2 in /opt/conda/lib/python3.10/site-pac
kages (from beautifulsoup4>=4.11.1->yfinance) (2.5)
Requirement already satisfied: python-dateutil>=2.8.2 in /opt/conda/lib/python3.10
/site-packages (from pandas>=1.3.0->yfinance) (2.9.0.post0)
Requirement already satisfied: tzdata>=2022.7 in /opt/conda/lib/python3.10/site-pa
ckages (from pandas>=1.3.0->yfinance) (2023.4)
Requirement already satisfied: charset-normalizer<4,>=2 in /opt/conda/lib/python3.
10/site-packages (from requests>=2.31->yfinance) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /opt/conda/lib/python3.10/site-pack
ages (from requests>=2.31->yfinance) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in /opt/conda/lib/python3.10/sit
e-packages (from requests>=2.31->yfinance) (1.26.18)
Requirement already satisfied: certifi>=2017.4.17 in /opt/conda/lib/python3.10/sit
e-packages (from requests>=2.31->yfinance) (2024.2.2)
Requirement already satisfied: six>=1.5 in /opt/conda/lib/python3.10/site-packages
 (from python-dateutil>=2.8.2->pandas>=1.3.0->yfinance) (1.16.0)
Downloading yfinance-0.2.54-py2.py3-none-any.whl (108 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 108.7/108.7 kB 5.1 MB/s eta 0:00:00
ultitasking-0.0.11-py3-none-any.whl (8.5 kB)
Building wheels for collected packages: peewee
  Building wheel for peewee (pyproject.toml) ... e=peewee-3.17.9-cp310-cp310-linux
_x86_64.whl size=317951 sha256=4be27a334f9bfa35baf5b962a5c8d85cc6c51e64144cecc4acc
f7a93607dec38
  Stored in directory: /root/.cache/pip/wheels/fd/fd/5e/90b9ec95da4fd6c96237b580ce
74f89d6bdea547ad151ab5f4
Successfully built peewee
Installing collected packages: peewee, multitasking, yfinance
Successfully installed multitasking-0.0.11 peewee-3.17.9 yfinance-0.2.54
```

```
#
========================================================================
=========
# Step 1: Import Libraries and Set Seeds for Reproducibility
#
========================================================================
=========

import os
import datetime
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
import tensorflow as tf
import yfinance as yf

from tensorflow.keras.models import Model, load_model
from tensorflow.keras.layers import (Dense, LSTM, Bidirectional,
Dropout, Input,
                                     Embedding, Flatten,
Concatenate, Reshape)
from tensorflow.keras.callbacks import EarlyStopping,
ModelCheckpoint, TensorBoard
from sklearn.preprocessing import MinMaxScaler, LabelEncoder
from sklearn.metrics import mean_squared_error

import warnings
warnings.filterwarnings('ignore')

# Set random seeds
np.random.seed(42)
tf.random.set_seed(42)

#
========================================================================
=========
# Step 2: Define Tickers, Sectors, and Download Data
#
========================================================================
=========

# Define at least 5 stocks per sector
# Tech, Energy, Finance, Auto, Retail
tickers = [
    # Tech
    "AAPL", "MSFT", "GOOG", "META", "NVDA",
    # Energy
    "XOM", "CVX", "BP", "COP", "EOG",
    # Finance
    "JPM", "BAC", "WFC", "C", "GS",
    # Auto
    "TSLA", "F", "GM", "HMC", "NIO",
```

```
    # Retail
    "AMZN", "TGT", "WMT", "COST", "HD"
]

sectors = {
    # Tech
    "AAPL": "Tech", "MSFT": "Tech", "GOOG": "Tech", "META": "Tech",
"NVDA": "Tech",
    # Energy
    "XOM": "Energy", "CVX": "Energy", "BP": "Energy", "COP":
"Energy", "EOG": "Energy",
    # Finance
    "JPM": "Finance", "BAC": "Finance", "WFC": "Finance", "C":
"Finance", "GS": "Finance",
    # Auto
    "TSLA": "Auto", "F": "Auto", "GM": "Auto", "HMC": "Auto", "NIO":
"Auto",
    # Retail
    "AMZN": "Retail", "TGT": "Retail", "WMT": "Retail", "COST":
"Retail", "HD": "Retail"
}

# Define date range
start_date = "2014-01-01"
end_date = "2024-01-01"
time_steps = 60

# Download 'Close' prices for all tickers in one call
data = yf.download(tickers, start=start_date, end=end_date)['Close']
data = data.reset_index()

# Melt data into long format and add metadata
df = data.melt(id_vars=['Date'], var_name='Ticker',
value_name='Close')
df['Sector'] = df['Ticker'].map(sectors)
df = df.sort_values(['Ticker', 'Date']).reset_index(drop=True)


# Drop rows with missing values (important!)
df.dropna(inplace=True)


min_count = df.groupby('Ticker')['Close'].count().min()
if min_count < time_steps:
    print("Warning: Some tickers have less than {} data
points.".format(time_steps))


#
====================================================================
=========
# Step 3: Encode Metadata and Scale Prices per Stock
#
```

```
=====================================================================
=========

# Create numeric IDs for embeddings
df['StockID'] = df['Ticker'].astype('category').cat.codes
df['SectorID'] = df['Sector'].astype('category').cat.codes

# Scale prices separately for each ticker to [0,1]
scalers = {}
for ticker in tickers:
    scaler = MinMaxScaler(feature_range=(0, 1))
    df.loc[df['Ticker'] == ticker, 'Scaled'] = scaler.fit_transform(
        df[df['Ticker'] == ticker][['Close']].values
    )
    scalers[ticker] = scaler  # store each scaler for inverse
transform later

#
=====================================================================
=========
# Step 4: Create Universal Sequences (with Price, StockID, and
SectorID)
#
=====================================================================
=========

def create_universal_sequences(df, time_steps=60):
    X_seq, y, stock_ids, sector_ids = [], [], [], []
    for ticker in df['Ticker'].unique():
        ticker_data = df[df['Ticker'] == ticker].sort_values('Date')
        scaled_prices = ticker_data['Scaled'].values
        stock_id = ticker_data['StockID'].iloc[0]
        sector_id = ticker_data['SectorID'].iloc[0]
        for i in range(len(scaled_prices) - time_steps):
            X_seq.append(scaled_prices[i:i+time_steps])
            y.append(scaled_prices[i+time_steps])
            stock_ids.append(stock_id)
            sector_ids.append(sector_id)
    return np.array(X_seq), np.array(y), np.array(stock_ids),
np.array(sector_ids)

time_steps = 60
X, y, stock_ids, sector_ids = create_universal_sequences(df,
time_steps)
X = X.reshape(X.shape[0], time_steps, 1)  # add feature dimension

#
=====================================================================
=========
# Step 5: Build Universal Models: One with LSTM and One with BiLSTM
#
=====================================================================
=========
```

```
def build_universal_model_lstm(time_steps, n_stocks, n_sectors):
    # Inputs for metadata and sequence
    stock_input = Input(shape=(1,), name='stock_input')
    sector_input = Input(shape=(1,), name='sector_input')
    price_input = Input(shape=(time_steps, 1), name='price_input')

    # Embedding layers for metadata
    stock_embed = Embedding(input_dim=n_stocks,
output_dim=8)(stock_input)
    sector_embed = Embedding(input_dim=n_sectors,
output_dim=4)(sector_input)
    stock_embed = Reshape((8,))(stock_embed)
    sector_embed = Reshape((4,))(sector_embed)

    # LSTM branch
    x = LSTM(128, return_sequences=True)(price_input)
    x = Dropout(0.2)(x)
    x = LSTM(64)(x)
    x = Dropout(0.2)(x)

    # Combine LSTM output with embeddings
    combined = Concatenate()([x, stock_embed, sector_embed])
    combined = Dense(32, activation='relu')(combined)
    output = Dense(1)(combined)

    model = Model(inputs=[price_input, stock_input, sector_input],
outputs=output)
    model.compile(optimizer='adam', loss='mse')
    return model

def build_universal_model_bilstm(time_steps, n_stocks, n_sectors):
    # Inputs for metadata and sequence
    stock_input = Input(shape=(1,), name='stock_input')
    sector_input = Input(shape=(1,), name='sector_input')
    price_input = Input(shape=(time_steps, 1), name='price_input')

    # Embedding layers for metadata
    stock_embed = Embedding(input_dim=n_stocks,
output_dim=8)(stock_input)
    sector_embed = Embedding(input_dim=n_sectors,
output_dim=4)(sector_input)
    stock_embed = Reshape((8,))(stock_embed)
    sector_embed = Reshape((4,))(sector_embed)

    # Bidirectional LSTM branch
    x = Bidirectional(LSTM(128, return_sequences=True))(price_input)
    x = Dropout(0.2)(x)
    x = Bidirectional(LSTM(64))(x)
    x = Dropout(0.2)(x)

    # Combine BiLSTM output with embeddings
    combined = Concatenate()([x, stock_embed, sector_embed])
```

```
    combined = Dense(32, activation='relu')(combined)
    output = Dense(1)(combined)

    model = Model(inputs=[price_input, stock_input, sector_input],
outputs=output)
    model.compile(optimizer='adam', loss='mse')
    return model

n_stocks = df['StockID'].nunique()
n_sectors = df['SectorID'].nunique()

universal_model_lstm = build_universal_model_lstm(time_steps,
n_stocks, n_sectors)
universal_model_bilstm = build_universal_model_bilstm(time_steps,
n_stocks, n_sectors)

print("--------------------------------------------------------------------
----------------")
print("Universal LSTM Model Summary:")
universal_model_lstm.summary()

print("--------------------------------------------------------------------
----------------")
print("\nUniversal BiLSTM Model Summary:")
universal_model_bilstm.summary()

#
====================================================================
=========
# Step 6: Split Data into Training and Testing Sets
#
====================================================================
=========

split_idx = int(0.8 * len(X))
X_train = [X[:split_idx], stock_ids[:split_idx],
sector_ids[:split_idx]]
y_train = y[:split_idx]
X_test = [X[split_idx:], stock_ids[split_idx:],
sector_ids[split_idx:]]
y_test = y[split_idx:]

#
====================================================================
=========
# Step 7: Set Up Callbacks
#
====================================================================
=========

model_dir = 'models'
os.makedirs(model_dir, exist_ok=True)
```

```
checkpoint_lstm_cb = ModelCheckpoint(os.path.join(model_dir,
'universal_lstm_best.keras'),
                                     monitor='val_loss',
save_best_only=True, verbose=1)
checkpoint_bilstm_cb = ModelCheckpoint(os.path.join(model_dir,
'universal_bilstm_best.keras'),
                                       monitor='val_loss',
save_best_only=True, verbose=1)

early_stop = EarlyStopping(monitor='val_loss', patience=10,
restore_best_weights=True, verbose=1)
log_dir = os.path.join("logs",
datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
tensorboard_callback = TensorBoard(log_dir=log_dir,
histogram_freq=1)

#
====================================================================
=========
# Step 8: Train Both Models
#
====================================================================
=========

print("-------------------------------------------------------------
----------------")
print("\nTraining Universal LSTM Model...")
history_lstm = universal_model_lstm.fit(
    X_train, y_train,
    validation_data=(X_test, y_test),
    epochs=50,
    batch_size=64,
    callbacks=[early_stop, checkpoint_lstm_cb,
tensorboard_callback],
    verbose=1
)

print("-------------------------------------------------------------
----------------")
print("\nTraining Universal BiLSTM Model...")
history_bilstm = universal_model_bilstm.fit(
    X_train, y_train,
    validation_data=(X_test, y_test),
    epochs=50,
    batch_size=64,
    callbacks=[early_stop, checkpoint_bilstm_cb,
tensorboard_callback],
    verbose=1
)

#
====================================================================
=========
```

```
# Step 9: Load Best Models and Evaluate on Test Set
#
=====================================================================
=========

universal_model_lstm = load_model(os.path.join(model_dir,
'universal_lstm_best.keras'))
universal_model_bilstm = load_model(os.path.join(model_dir,
'universal_bilstm_best.keras'))

# Predictions from each model
preds_lstm = universal_model_lstm.predict(X_test)
preds_bilstm = universal_model_bilstm.predict(X_test)

# Ensemble: average predictions from both models
ensemble_preds = (preds_lstm + preds_bilstm) / 2.0

# Inverse scale predictions per sample using the corresponding
scaler (per ticker)
def inverse_scale_predictions(preds, X_stock_ids, y_true):
    final_preds = []
    final_y = []
    for i, (pred, stock_id_val) in enumerate(zip(preds,
X_stock_ids)):
        # Get the ticker name from the categorical mapping
        ticker_name =
df['Ticker'].astype('category').cat.categories[stock_id_val]
        scaler = scalers[ticker_name]
        # Inverse transform a single prediction
        pred_inv = scaler.inverse_transform(np.array([[pred[0]]]))
        final_preds.append(pred_inv[0][0])

        # Also inverse transform the true value
        y_inv = scaler.inverse_transform(np.array([[y_true[i]]]))
        final_y.append(y_inv[0][0])
    return np.array(final_preds), np.array(final_y)

# For each model, do the inverse transformation
lstm_preds_final, y_test_final =
inverse_scale_predictions(preds_lstm, X_test[1], y_test)
bilstm_preds_final, _ = inverse_scale_predictions(preds_bilstm,
X_test[1], y_test)
ensemble_preds_final, _ = inverse_scale_predictions(ensemble_preds,
X_test[1], y_test)

def calculate_rmse(actual, predicted):
    return math.sqrt(mean_squared_error(actual, predicted))

rmse_lstm = calculate_rmse(y_test_final, lstm_preds_final)
rmse_bilstm = calculate_rmse(y_test_final, bilstm_preds_final)
rmse_ensemble = calculate_rmse(y_test_final, ensemble_preds_final)

print("-------------------------------------------------------------
```

```
----------------")
print(f"\nUniversal LSTM Model RMSE: {rmse_lstm:.4f}")
print(f"Universal BiLSTM Model RMSE: {rmse_bilstm:.4f}")
print(f"Ensemble Model RMSE: {rmse_ensemble:.4f}")
print("------------------------------------------------------------------
----------------")

#
===================================================================
=========
# Step 10: Visualization - Plot Predictions per Ticker
#
===================================================================
=========

# Get unique tickers from test set stock IDs
unique_stock_ids = np.unique(X_test[1])
categories = df['Ticker'].astype('category').cat.categories

plt.figure(figsize=(18, 12))
plot_idx = 1
for stock_id in unique_stock_ids:
    ticker_name = categories[stock_id]
    # Find indices in test set corresponding to this ticker
    mask = (X_test[1] == stock_id)
    if np.sum(mask) == 0:
        continue
    actual = y_test_final[mask]
    pred_lstm = lstm_preds_final[mask]
    pred_bilstm = bilstm_preds_final[mask]
    pred_ensemble = ensemble_preds_final[mask]

    plt.subplot(3, 3, plot_idx)
    plt.plot(actual, label='Actual', color='blue')
    plt.plot(pred_lstm, label='LSTM', linestyle='--', color='red')
    plt.plot(pred_bilstm, label='BiLSTM', linestyle='--',
color='green')
    plt.plot(pred_ensemble, label='Ensemble', linestyle='-.',
color='purple')
    plt.title(f'{ticker_name} Price Predictions')
    plt.xlabel('Time Steps')
    plt.ylabel('Price')
    plt.legend()
    plot_idx += 1

plt.tight_layout()
plt.show()
```

```
2025-02-22 23:57:53.961095: E external/local_xla/xla/stream_executor/cuda/
cuda_dnn.cc:9261] Unable to register cuDNN factory: Attempting to register
 factory for plugin cuDNN when one has already been registered
2025-02-22 23:57:53.961243: E external/local_xla/xla/stream_executor/cuda/
```

```
cuda_fft.cc:607] Unable to register cuFFT factory: Attempting to register factory for plugin cuFFT when one has already been registered
2025-02-22 23:57:54.218202: E external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1515] Unable to register cuBLAS factory: Attempting to register factory for plugin cuBLAS when one has already been registered

YF.download() has changed argument auto_adjust default to True

[*******************100%**********************]  25 of 25 completed

----------------------------------------------------------------------
Universal LSTM Model Summary:

Model: "functional_1"
```

Layer (type)	Output Shape	Param #	Connected to
price_input (InputLayer)	(None, 60, 1)	0	-
lstm (LSTM)	(None, 60, 128)	66,560	price_input[0][0]
dropout (Dropout)	(None, 60, 128)	0	lstm[0][0]
stock_input (InputLayer)	(None, 1)	0	-
sector_input (InputLayer)	(None, 1)	0	-
lstm_1 (LSTM)	(None, 64)	49,408	dropout[0][0]
embedding (Embedding)	(None, 1, 8)	200	stock_input[0][0]
embedding_1 (Embedding)	(None, 1, 4)	20	sector_input[0][…
dropout_1 (Dropout)	(None, 64)	0	lstm_1[0][0]
reshape (Reshape)	(None, 8)	0	embedding[0][0]
reshape_1 (Reshape)	(None, 4)	0	embedding_1[0][0]
concatenate (Concatenate)	(None, 76)	0	dropout_1[0][0], reshape[0][0], reshape_1[0][0]
dense (Dense)	(None, 32)	2,464	concatenate[0][0]
dense_1 (Dense)	(None, 1)	33	dense[0][0]

```
 Total params: 118,685 (463.61 KB)

 Trainable params: 118,685 (463.61 KB)
```

```
 Non-trainable params: 0 (0.00 B)

-----------------------------------------------------------------------
---

Universal BiLSTM Model Summary:

Model: "functional_3"
```

Layer (type)	Output Shape	Param #	Connected to
price_input (InputLayer)	(None, 60, 1)	0	-
bidirectional (Bidirectional)	(None, 60, 256)	133,120	price_input[0][0]
dropout_2 (Dropout)	(None, 60, 256)	0	bidirectional[0]…
stock_input (InputLayer)	(None, 1)	0	-
sector_input (InputLayer)	(None, 1)	0	-
bidirectional_1 (Bidirectional)	(None, 128)	164,352	dropout_2[0][0]
embedding_2 (Embedding)	(None, 1, 8)	200	stock_input[0][0]
embedding_3 (Embedding)	(None, 1, 4)	20	sector_input[0][…
dropout_3 (Dropout)	(None, 128)	0	bidirectional_1[…
reshape_2 (Reshape)	(None, 8)	0	embedding_2[0][0]
reshape_3 (Reshape)	(None, 4)	0	embedding_3[0][0]
concatenate_1 (Concatenate)	(None, 140)	0	dropout_3[0][0], reshape_2[0][0], reshape_3[0][0]
dense_2 (Dense)	(None, 32)	4,512	concatenate_1[0]…
dense_3 (Dense)	(None, 1)	33	dense_2[0][0]

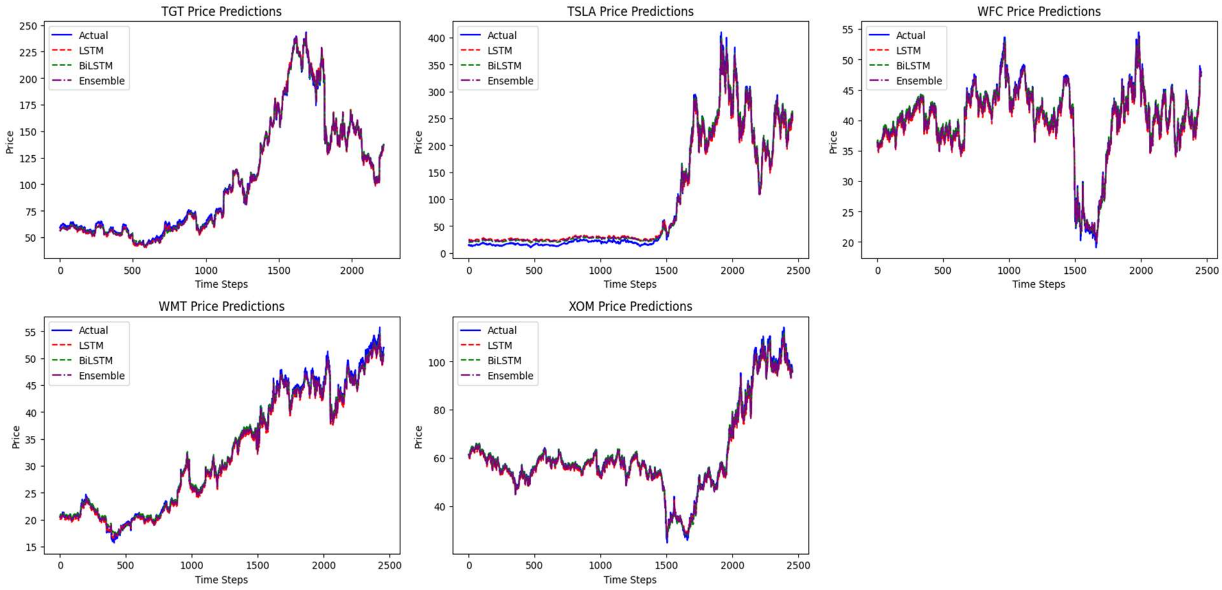
```
 Total params: 302,237 (1.15 MB)

 Trainable params: 302,237 (1.15 MB)

 Non-trainable params: 0 (0.00 B)

-----------------------------------------------------------------------
---
```

```
Training Universal LSTM Model...
Epoch 1/50
749/753 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0104
Epoch 1: val_loss improved from inf to 0.00078, saving model to models/uni
versal_lstm_best.keras
753/753 ━━━━━━━━━━━━━━━━━━━━ 13s 10ms/step - loss: 0.0104 - val_loss: 7.81
13e-04
Epoch 2/50
748/753 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 0.0011
Epoch 2: val_loss did not improve from 0.00078
753/753 ━━━━━━━━━━━━━━━━━━━━ 7s 9ms/step - loss: 0.0011 - val_loss: 0.0027
Epoch 3/50
753/753 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 7.5445e-04
Epoch 3: val_loss did not improve from 0.00078
753/753 ━━━━━━━━━━━━━━━━━━━━ 7s 9ms/step - loss: 7.5438e-04 - val_loss: 0.
0019
Epoch 4/50
750/753 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 6.3584e-04
Epoch 4: val_loss did not improve from 0.00078
753/753 ━━━━━━━━━━━━━━━━━━━━ 7s 9ms/step - loss: 6.3569e-04 - val_loss: 0.
0021
Epoch 5/50
753/753 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 5.6462e-04
Epoch 5: val_loss did not improve from 0.00078
753/753 ━━━━━━━━━━━━━━━━━━━━ 7s 9ms/step - loss: 5.6458e-04 - val_loss: 0.
0014
Epoch 6/50
752/753 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 5.1693e-04
Epoch 6: val_loss did not improve from 0.00078
753/753 ━━━━━━━━━━━━━━━━━━━━ 7s 9ms/step - loss: 5.1687e-04 - val_loss: 0.
0012
Epoch 7/50
749/753 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 4.9497e-04
Epoch 7: val_loss did not improve from 0.00078
753/753 ━━━━━━━━━━━━━━━━━━━━ 7s 9ms/step - loss: 4.9483e-04 - val_loss: 8.
4871e-04
Epoch 8/50
750/753 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 4.6889e-04
Epoch 8: val_loss did not improve from 0.00078
753/753 ━━━━━━━━━━━━━━━━━━━━ 7s 9ms/step - loss: 4.6885e-04 - val_loss: 0.
0010
Epoch 9/50
752/753 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 4.4571e-04
Epoch 9: val_loss improved from 0.00078 to 0.00048, saving model to models
/universal_lstm_best.keras
753/753 ━━━━━━━━━━━━━━━━━━━━ 7s 9ms/step - loss: 4.4570e-04 - val_loss: 4.
7883e-04
Epoch 10/50
749/753 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 4.6200e-04
Epoch 10: val_loss did not improve from 0.00048
753/753 ━━━━━━━━━━━━━━━━━━━━ 7s 9ms/step - loss: 4.6184e-04 - val_loss: 6.
0829e-04
Epoch 11/50
753/753 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 4.3172e-04
```

```
Epoch 11: val_loss did not improve from 0.00048
753/753 ━━━━━━━━━━━━━━━━━━━━ 7s 9ms/step - loss: 4.3171e-04 - val_loss: 6.
1946e-04
Epoch 12/50
752/753 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 4.1476e-04
Epoch 12: val_loss did not improve from 0.00048
753/753 ━━━━━━━━━━━━━━━━━━━━ 7s 9ms/step - loss: 4.1471e-04 - val_loss: 7.
4891e-04
Epoch 13/50
752/753 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 3.9123e-04
Epoch 13: val_loss did not improve from 0.00048
753/753 ━━━━━━━━━━━━━━━━━━━━ 7s 9ms/step - loss: 3.9119e-04 - val_loss: 7.
1975e-04
Epoch 14/50
752/753 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 3.7722e-04
Epoch 14: val_loss did not improve from 0.00048
753/753 ━━━━━━━━━━━━━━━━━━━━ 7s 9ms/step - loss: 3.7716e-04 - val_loss: 0.
0011
Epoch 15/50
747/753 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 3.4747e-04
Epoch 15: val_loss did not improve from 0.00048
753/753 ━━━━━━━━━━━━━━━━━━━━ 7s 9ms/step - loss: 3.4736e-04 - val_loss: 0.
0012
Epoch 16/50
753/753 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 3.4018e-04
Epoch 16: val_loss did not improve from 0.00048
753/753 ━━━━━━━━━━━━━━━━━━━━ 7s 9ms/step - loss: 3.4016e-04 - val_loss: 0.
0012
Epoch 17/50
752/753 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 3.3203e-04
Epoch 17: val_loss did not improve from 0.00048
753/753 ━━━━━━━━━━━━━━━━━━━━ 7s 9ms/step - loss: 3.3199e-04 - val_loss: 0.
0015
Epoch 18/50
751/753 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 3.1488e-04
Epoch 18: val_loss did not improve from 0.00048
753/753 ━━━━━━━━━━━━━━━━━━━━ 7s 9ms/step - loss: 3.1486e-04 - val_loss: 0.
0017
Epoch 19/50
749/753 ━━━━━━━━━━━━━━━━━━━━ 0s 8ms/step - loss: 3.0630e-04
Epoch 19: val_loss did not improve from 0.00048
753/753 ━━━━━━━━━━━━━━━━━━━━ 7s 9ms/step - loss: 3.0626e-04 - val_loss: 0.
0016
Epoch 19: early stopping
Restoring model weights from the end of the best epoch: 9.
--------------------------------------------------------------------------
---

Training Universal BiLSTM Model...
Epoch 1/50
752/753 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 0.0076
Epoch 1: val_loss improved from inf to 0.00045, saving model to models/uni
versal_bilstm_best.keras
753/753 ━━━━━━━━━━━━━━━━━━━━ 16s 16ms/step - loss: 0.0075 - val_loss: 4.51
```

```
23e-04
Epoch 2/50
753/753 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 7.9383e-04
Epoch 2: val_loss improved from 0.00045 to 0.00039, saving model to models
/universal_bilstm_best.keras
753/753 ━━━━━━━━━━━━━━━━━━━━ 12s 15ms/step - loss: 7.9374e-04 - val_loss:
3.9481e-04
Epoch 3/50
753/753 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 6.3627e-04
Epoch 3: val_loss did not improve from 0.00039
753/753 ━━━━━━━━━━━━━━━━━━━━ 12s 15ms/step - loss: 6.3622e-04 - val_loss:
4.0234e-04
Epoch 4/50
753/753 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 5.6118e-04
Epoch 4: val_loss improved from 0.00039 to 0.00032, saving model to models
/universal_bilstm_best.keras
753/753 ━━━━━━━━━━━━━━━━━━━━ 12s 16ms/step - loss: 5.6114e-04 - val_loss:
3.1714e-04
Epoch 5/50
753/753 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 5.2890e-04
Epoch 5: val_loss did not improve from 0.00032
753/753 ━━━━━━━━━━━━━━━━━━━━ 12s 16ms/step - loss: 5.2888e-04 - val_loss:
4.1247e-04
Epoch 6/50
752/753 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 5.1006e-04
Epoch 6: val_loss did not improve from 0.00032
753/753 ━━━━━━━━━━━━━━━━━━━━ 12s 15ms/step - loss: 5.1004e-04 - val_loss:
3.3192e-04
Epoch 7/50
751/753 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 4.7339e-04
Epoch 7: val_loss did not improve from 0.00032
753/753 ━━━━━━━━━━━━━━━━━━━━ 12s 15ms/step - loss: 4.7339e-04 - val_loss:
3.4221e-04
Epoch 8/50
753/753 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 4.7799e-04
Epoch 8: val_loss did not improve from 0.00032
753/753 ━━━━━━━━━━━━━━━━━━━━ 12s 15ms/step - loss: 4.7797e-04 - val_loss:
4.1498e-04
Epoch 9/50
753/753 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 4.8528e-04
Epoch 9: val_loss improved from 0.00032 to 0.00030, saving model to models
/universal_bilstm_best.keras
753/753 ━━━━━━━━━━━━━━━━━━━━ 12s 16ms/step - loss: 4.8525e-04 - val_loss:
3.0258e-04
Epoch 10/50
752/753 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 4.4352e-04
Epoch 10: val_loss did not improve from 0.00030
753/753 ━━━━━━━━━━━━━━━━━━━━ 12s 15ms/step - loss: 4.4353e-04 - val_loss:
3.1705e-04
Epoch 11/50
752/753 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 4.2442e-04
Epoch 11: val_loss did not improve from 0.00030
753/753 ━━━━━━━━━━━━━━━━━━━━ 12s 15ms/step - loss: 4.2441e-04 - val_loss:
4.3797e-04
```

```
Epoch 12/50
751/753 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 4.4444e-04
Epoch 12: val_loss did not improve from 0.00030
753/753 ━━━━━━━━━━━━━━━━━━━━ 12s 16ms/step - loss: 4.4433e-04 - val_loss:
4.7283e-04
Epoch 13/50
752/753 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 3.9746e-04
Epoch 13: val_loss did not improve from 0.00030
753/753 ━━━━━━━━━━━━━━━━━━━━ 12s 15ms/step - loss: 3.9743e-04 - val_loss:
8.4394e-04
Epoch 14/50
752/753 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 3.8004e-04
Epoch 14: val_loss did not improve from 0.00030
753/753 ━━━━━━━━━━━━━━━━━━━━ 12s 15ms/step - loss: 3.8000e-04 - val_loss:
5.7543e-04
Epoch 15/50
750/753 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 3.7365e-04
Epoch 15: val_loss did not improve from 0.00030
753/753 ━━━━━━━━━━━━━━━━━━━━ 12s 16ms/step - loss: 3.7352e-04 - val_loss:
9.5113e-04
Epoch 16/50
753/753 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 3.3620e-04
Epoch 16: val_loss did not improve from 0.00030
753/753 ━━━━━━━━━━━━━━━━━━━━ 12s 15ms/step - loss: 3.3619e-04 - val_loss:
7.4471e-04
Epoch 17/50
752/753 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 3.3355e-04
Epoch 17: val_loss did not improve from 0.00030
753/753 ━━━━━━━━━━━━━━━━━━━━ 12s 16ms/step - loss: 3.3351e-04 - val_loss:
0.0011
Epoch 18/50
752/753 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 3.3622e-04
Epoch 18: val_loss did not improve from 0.00030
753/753 ━━━━━━━━━━━━━━━━━━━━ 12s 15ms/step - loss: 3.3618e-04 - val_loss:
0.0011
Epoch 19/50
751/753 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 3.1929e-04
Epoch 19: val_loss did not improve from 0.00030
753/753 ━━━━━━━━━━━━━━━━━━━━ 12s 15ms/step - loss: 3.1929e-04 - val_loss:
0.0012
Epoch 19: early stopping
Restoring model weights from the end of the best epoch: 9.
377/377 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
377/377 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step
--------------------------------------------------------------------------
---

Universal LSTM Model RMSE: 4.8670
Universal BiLSTM Model RMSE: 3.7643
Ensemble Model RMSE: 4.0699
--------------------------------------------------------------------------
---
```


TGT Price Predictions
TSLA Price Predictions
WFC Price Predictions
WMT Price Predictions
XOM Price Predictions
Actual
LSTM
BiLSTM
Ensemble
Price
Time Steps

TEST MODEL ON NEW STOCK PRICE DATA:

```
import os
import datetime
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
import tensorflow as tf
import yfinance as yf

from tensorflow.keras.models import Model, load_model
from tensorflow.keras.layers import Dense, LSTM, Bidirectional,
Dropout, Input, Embedding, Reshape, Concatenate
from tensorflow.keras.callbacks import EarlyStopping,
ModelCheckpoint, TensorBoard
from tensorflow.keras.optimizers import Adam
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
import warnings
warnings.filterwarnings('ignore')

np.random.seed(42)
tf.random.set_seed(42)

# ------------------------------------------------------------------
-----------
# Assume these definitions are already in your training code:
# ------------------------------------------------------------------
-----------

time_steps = 60
old_n_stocks = 25    # Number of stocks in your original training
data
n_sectors = 5        # Assume the sector count remains the same

def build_universal_model_bidirectional(time_steps, n_stocks,
n_sectors):
    # Inputs
    price_input = Input(shape=(time_steps, 1), name='price_input')
    stock_input = Input(shape=(1,), name='stock_input')
    sector_input = Input(shape=(1,), name='sector_input')

    # Embedding layers with explicit names
    stock_embed = Embedding(input_dim=n_stocks, output_dim=8,
name='embedding_2')(stock_input)
    sector_embed = Embedding(input_dim=n_sectors, output_dim=4,
name='embedding_3')(sector_input)
    stock_embed = Reshape((8,), name='reshape_2')(stock_embed)
    sector_embed = Reshape((4,), name='reshape_3')(sector_embed)
```

```
    # Bidirectional LSTM branch
    x = Bidirectional(LSTM(128, return_sequences=True),
name='bidirectional')(price_input)
    x = Dropout(0.2, name='dropout_2')(x)
    x = Bidirectional(LSTM(64), name='bidirectional_1')(x)
    x = Dropout(0.2, name='dropout_3')(x)

    # Combine LSTM output with embeddings
    combined = Concatenate(name='concatenate_1')([x, stock_embed,
sector_embed])
    x = Dense(32, activation='relu', name='dense_2')(combined)
    output = Dense(1, name='dense_3')(x)

    optimizer = Adam(learning_rate=0.001, clipvalue=1.0)
    model = Model(inputs=[price_input, stock_input, sector_input],
outputs=output)
    model.compile(optimizer=optimizer, loss='mse')
    return model

def create_universal_sequences(df, time_steps=60):
    X_seq, y_seq, stock_ids, sector_ids = [], [], [], []
    for ticker in df['Ticker'].unique():
        ticker_data = df[df['Ticker'] == ticker].sort_values('Date')
        scaled_prices = ticker_data['Scaled'].values
        stock_id = ticker_data['StockID'].iloc[0]
        sector_id = ticker_data['SectorID'].iloc[0]
        for i in range(len(scaled_prices) - time_steps):
            X_seq.append(scaled_prices[i:i+time_steps])
            y_seq.append(scaled_prices[i+time_steps])
            stock_ids.append(stock_id)
            sector_ids.append(sector_id)
    return np.array(X_seq), np.array(y_seq), np.array(stock_ids),
np.array(sector_ids)

# ------------------------------------------------------------------
-----------
# Load your pre-trained universal model (from training)
# ------------------------------------------------------------------
-----------
# Here we load the bidirectional model that was saved (assume
'universal_bilstm_best.keras')
old_model = load_model(os.path.join("models",
"universal_bilstm_best.keras"))

# ------------------------------------------------------------------
-----------
# Expand the model to include the new ticker by increasing the
embedding dimension
# ------------------------------------------------------------------
-----------
def expand_model_for_new_ticker(old_model, old_n_stocks,
new_ticker):
    new_n_stocks = old_n_stocks + 1  # add one new ticker
```

```
    new_model = build_universal_model_bidirectional(time_steps,
new_n_stocks, n_sectors)

    # Transfer weights from old model for all layers except the
stock embedding
    old_stock_embed_layer = old_model.get_layer("embedding_2")
    old_stock_embed_weights = old_stock_embed_layer.get_weights()[0]
# shape (old_n_stocks, 8)

    new_stock_embed_layer = new_model.get_layer("embedding_2")
    new_embed_shape = new_stock_embed_layer.get_weights()[0].shape
# shape (new_n_stocks, 8)

    # Create new embedding weights: copy old weights and initialize
new row for the new ticker
    new_stock_embed_weights = np.zeros(new_embed_shape)
    new_stock_embed_weights[:old_n_stocks, :] =
old_stock_embed_weights
    new_stock_embed_weights[old_n_stocks, :] =
np.mean(old_stock_embed_weights, axis=0)
    new_stock_embed_layer.set_weights([new_stock_embed_weights])

    # Transfer weights for all other layers
    for layer in new_model.layers:
        if layer.name not in ["embedding_2"]:
            try:
                old_layer = old_model.get_layer(layer.name)
                layer.set_weights(old_layer.get_weights())
            except Exception as e:
                print(f"Could not transfer weights for layer
{layer.name}: {e}")
    return new_model

# -------------------------------------------------------------------
-----------
# Define a function to take user input and predict prices for a new
ticker
# -------------------------------------------------------------------
-----------
def predict_new_stock():
    # Get user inputs
    user_ticker = input("Enter the stock ticker for prediction
(e.g., IBM): ").strip().upper()
    user_sector = input("Enter the stock sector (e.g., Tech, Energy,
Finance, Auto, Retail): ").strip()
    start_date_input = input("Enter the start date for historical
data (YYYY-MM-DD): ").strip()
    end_date_input = input("Enter the end date for historical data
(YYYY-MM-DD): ").strip()

    # Download historical data for the user-specified ticker
    new_data = yf.download(user_ticker, start=start_date_input,
end=end_date_input)
```

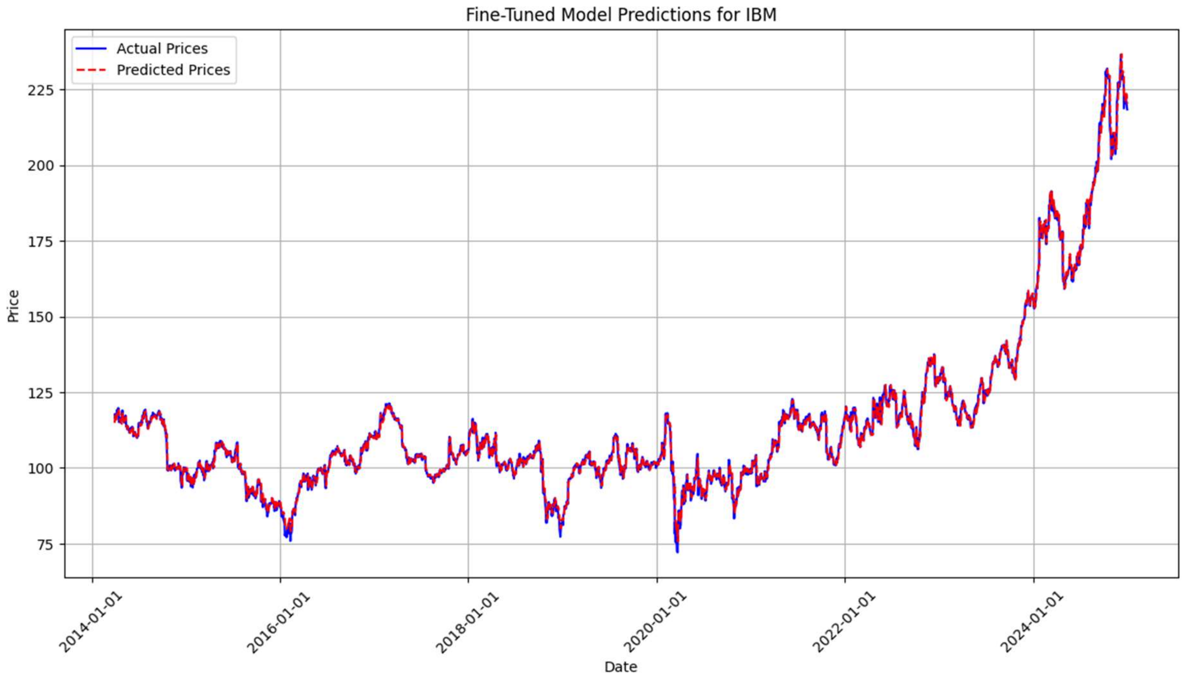
```
    if new_data.empty:
        print("No data found for this ticker in the given date
range.")
        return
    new_data = new_data[['Close']].reset_index()

    # Create a DataFrame with metadata
    new_df = new_data.copy()
    new_df["Ticker"] = user_ticker
    new_df["Sector"] = user_sector
    new_df = new_df.sort_values("Date").reset_index(drop=True)

    # Scale the data using a new scaler for this ticker
    scaler_new = MinMaxScaler(feature_range=(0, 1))
    new_df["Scaled"] = scaler_new.fit_transform(new_df[["Close"]])

    # For a new ticker, assign a new StockID = old_n_stocks (since 0
to old_n_stocks-1 are taken)
    new_stock_id = old_n_stocks
    # For sector, if the user-specified sector is one of the known
ones, assign its ID; else, default to 0.
    known_sectors = ["Tech", "Energy", "Finance", "Auto", "Retail"]
    if user_sector in known_sectors:
        new_sector_id = known_sectors.index(user_sector)
    else:
        new_sector_id = 0
        print("Warning: The entered sector is not recognized;
defaulting to sector ID 0.")

    new_df["StockID"] = new_stock_id
    new_df["SectorID"] = new_sector_id

    # Create sequences from the new data
    X_new_seq, y_new, _, _ = create_universal_sequences(new_df,
time_steps)
    X_new_seq = X_new_seq.reshape(X_new_seq.shape[0], time_steps, 1)

    # Prepare inputs for prediction: for all samples, stock_input
and sector_input are constant
    num_samples = X_new_seq.shape[0]
    X_new_stock = np.full((num_samples, 1), new_stock_id)
    X_new_sector = np.full((num_samples, 1), new_sector_id)

    # Expand the pre-trained model to include the new ticker
    new_model = expand_model_for_new_ticker(old_model, old_n_stocks,
user_ticker)

    # Fine-tune the new model on the new ticker's data
    fine_tune_lr = 1e-4
    new_model.compile(optimizer=Adam(learning_rate=fine_tune_lr,
clipvalue=1.0), loss='mse')
    fine_tune_callbacks = [EarlyStopping(monitor='loss', patience=3,
restore_best_weights=True, verbose=1)]
```

```
    new_model.fit(
        [X_new_seq, X_new_stock, X_new_sector], y_new,
        epochs=20,
        batch_size=32,
        callbacks=fine_tune_callbacks,
        verbose=1
    )

    # Predict using the fine-tuned model
    preds_new_scaled = new_model.predict([X_new_seq, X_new_stock,
X_new_sector])
    preds_new = scaler_new.inverse_transform(preds_new_scaled)
    y_new_actual = scaler_new.inverse_transform(y_new.reshape(-1,
1))

    rmse_new = math.sqrt(mean_squared_error(y_new_actual,
preds_new))
    print("-------------------------------------------------------------
--------------------")
    print(f"Fine-Tuned Model RMSE on {user_ticker} Data:
{rmse_new:.4f}")
    print("-------------------------------------------------------------
--------------------")
    print("\n\n")

    # Extract target dates corresponding to each prediction (targets
are taken from new_df["Date"].iloc[time_steps:])
    target_dates =
new_df["Date"].iloc[time_steps:].reset_index(drop=True)

    import matplotlib.dates as mdates
    plt.figure(figsize=(14, 7))
    plt.plot(target_dates, y_new_actual, label="Actual Prices",
color="blue")
    plt.plot(target_dates, preds_new, label="Predicted Prices",
color="red", linestyle="--")

    plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-
%d'))
    plt.gca().xaxis.set_major_locator(mdates.AutoDateLocator())
    plt.xticks(rotation=45)
    plt.title(f"Fine-Tuned Model Predictions for {user_ticker}")
    plt.xlabel("Date")
    plt.ylabel("Price")
    plt.legend()
    plt.grid()
    plt.show()

# ------------------------------------------------------------------
-----------
# Call the function to let the user input a stock name and sector
# ------------------------------------------------------------------
```

```
-----------
predict_new_stock()
```

```
Enter the stock ticker for prediction (e.g., IBM):  IBM
Enter the stock sector (e.g., Tech, Energy, Finance, Auto, Retail):
Tech
Enter the start date for historical data (YYYY-MM-DD):  2014-01-01
Enter the end date for historical data (YYYY-MM-DD):  2025-01-01
```

```
[*********************100%***********************]  1 of 1 completed

Epoch 1/20


85/85 ━━━━━━━━━━━━━━━━━━━━ 4s 14ms/step - loss: 2.5906e-04
Epoch 2/20
85/85 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 2.3577e-04
Epoch 3/20
85/85 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - loss: 2.5483e-04
Epoch 4/20
85/85 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - loss: 2.3322e-04
Epoch 5/20
85/85 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - loss: 2.1628e-04
Epoch 6/20
85/85 ━━━━━━━━━━━━━━━━━━━━ 1s 13ms/step - loss: 2.2539e-04
Epoch 7/20
85/85 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - loss: 2.1908e-04
Epoch 8/20
85/85 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - loss: 2.1260e-04
Epoch 9/20
85/85 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - loss: 2.1687e-04
Epoch 10/20
85/85 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - loss: 2.0872e-04
Epoch 11/20
85/85 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - loss: 2.1087e-04
Epoch 12/20
85/85 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - loss: 2.0709e-04
Epoch 13/20
85/85 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - loss: 2.3104e-04
Epoch 14/20
85/85 ━━━━━━━━━━━━━━━━━━━━ 1s 12ms/step - loss: 2.0135e-04
Epoch 14: early stopping
Restoring model weights from the end of the best epoch: 11.
85/85 ━━━━━━━━━━━━━━━━━━━━ 1s 9ms/step
--------------------------------------------------------------------------
---
Fine-Tuned Model RMSE on IBM Data: 1.7317
--------------------------------------------------------------------------
---
```


Fine-Tuned Model Predictions for IBM
Actual Prices
Predicted Prices
Price
225
200
175
150
125
100
75
2014-01-01
2016-01-01
2018-01-01
2020-01-01
2022-01-01
2024-01-01
Date

FORECAST STOCK PRICES BY FINETUNING MODEL:

```
import os
import datetime
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import math
import tensorflow as tf
import yfinance as yf

from tensorflow.keras.models import Model, load_model
from tensorflow.keras.layers import Dense, LSTM, Bidirectional,
Dropout, Input, Embedding, Reshape, Concatenate
from tensorflow.keras.callbacks import EarlyStopping,
ModelCheckpoint, TensorBoard
from tensorflow.keras.optimizers import Adam
from sklearn.preprocessing import MinMaxScaler, LabelEncoder
from sklearn.metrics import mean_squared_error
import matplotlib.dates as mdates
import warnings
warnings.filterwarnings('ignore')

np.random.seed(42)
tf.random.set_seed(42)

# -------------------------
# Parameters and Pre-trained Model Info
# -------------------------
time_steps = 60
old_n_stocks = 25    # Number of stocks in your original training
data
n_sectors = 5        # Assume the sector count remains the same

# -------------------------
# Model Definition: Universal Bidirectional LSTM
# -------------------------
def build_universal_model_bidirectional(time_steps, n_stocks,
n_sectors):
    # Define inputs
    price_input = Input(shape=(time_steps, 1), name='price_input')
    stock_input = Input(shape=(1,), name='stock_input')
    sector_input = Input(shape=(1,), name='sector_input')

    # Embedding layers with explicit names
    stock_embed = Embedding(input_dim=n_stocks, output_dim=8,
name='embedding_2')(stock_input)
    sector_embed = Embedding(input_dim=n_sectors, output_dim=4,
name='embedding_3')(sector_input)
    stock_embed = Reshape((8,), name='reshape_2')(stock_embed)
    sector_embed = Reshape((4,), name='reshape_3')(sector_embed)
```

```
    # Bidirectional LSTM branch
    x = Bidirectional(LSTM(128, return_sequences=True),
name='bidirectional')(price_input)
    x = Dropout(0.2, name='dropout_2')(x)
    x = Bidirectional(LSTM(64), name='bidirectional_1')(x)
    x = Dropout(0.2, name='dropout_3')(x)

    # Combine LSTM output with embeddings
    combined = Concatenate(name='concatenate_1')([x, stock_embed,
sector_embed])
    x = Dense(32, activation='relu', name='dense_2')(combined)
    output = Dense(1, name='dense_3')(x)

    optimizer = Adam(learning_rate=0.001, clipvalue=1.0)
    model = Model(inputs=[price_input, stock_input, sector_input],
outputs=output)
    model.compile(optimizer=optimizer, loss='mse')
    return model

# -------------------------
# Sequence Creation Function
# -------------------------
def create_universal_sequences(df, time_steps=60):
    X_seq, y_seq, stock_ids, sector_ids = [], [], [], []
    for ticker in df['Ticker'].unique():
        ticker_data = df[df['Ticker'] == ticker].sort_values('Date')
        scaled_prices = ticker_data['Scaled'].values
        stock_id = ticker_data['StockID'].iloc[0]
        sector_id = ticker_data['SectorID'].iloc[0]
        for i in range(len(scaled_prices) - time_steps):
            X_seq.append(scaled_prices[i:i+time_steps])
            y_seq.append(scaled_prices[i+time_steps])
            stock_ids.append(stock_id)
            sector_ids.append(sector_id)
    return np.array(X_seq), np.array(y_seq), np.array(stock_ids),
np.array(sector_ids)

# -------------------------
# Model Expansion Function for New Ticker
# -------------------------
def expand_model_for_new_ticker(old_model, old_n_stocks,
new_ticker):
    new_n_stocks = old_n_stocks + 1  # expand the embedding
dimension
    new_model = build_universal_model_bidirectional(time_steps,
new_n_stocks, n_sectors)

    # Transfer weights for the stock embedding layer
    old_stock_embed_layer = old_model.get_layer("embedding_2")
    old_stock_embed_weights = old_stock_embed_layer.get_weights()[0]
# shape (old_n_stocks, 8)
```

```
    new_stock_embed_layer = new_model.get_layer("embedding_2")
    new_embed_shape = new_stock_embed_layer.get_weights()[0].shape
# shape (new_n_stocks, 8)

    new_stock_embed_weights = np.zeros(new_embed_shape)
    new_stock_embed_weights[:old_n_stocks, :] =
old_stock_embed_weights
    new_stock_embed_weights[old_n_stocks, :] =
np.mean(old_stock_embed_weights, axis=0)
    new_stock_embed_layer.set_weights([new_stock_embed_weights])

    # Transfer weights for other layers
    for layer in new_model.layers:
        if layer.name not in ["embedding_2"]:
            try:
                old_layer = old_model.get_layer(layer.name)
                layer.set_weights(old_layer.get_weights())
            except Exception as e:
                print(f"Could not transfer weights for layer
{layer.name}: {e}")
    return new_model

# -------------------------
# Iterative Forecasting Function
# -------------------------
def iterative_forecast(model, initial_sequence, stock_id, sector_id,
forecast_horizon):
    current_sequence = initial_sequence.copy()  # shape:
(time_steps, 1)
    forecasts = []
    for _ in range(forecast_horizon):
        seq_input = current_sequence.reshape(1, time_steps, 1)
        pred_scaled = model.predict([seq_input, stock_id,
sector_id])
        forecasts.append(pred_scaled[0, 0])
        current_sequence = np.append(current_sequence[1:],
[[pred_scaled[0, 0]]], axis=0)
    return forecasts

# -------------------------
# Load Pre-trained Universal Model
# -------------------------
# Assume your pre-trained bidirectional model is saved as
"models/universal_bilstm_best.keras"
old_model = load_model(os.path.join("models",
"universal_bilstm_best.keras"))

# -------------------------
# Main Function: User Input, Fine-Tuning, and Forecasting
# -------------------------
def predict_and_forecast_new_stock():
    # Get user inputs
    user_ticker = input("Enter the stock ticker for prediction
```

```
(e.g., IBM): ").strip().upper()
    user_sector = input("Enter the stock sector (e.g., Tech, Energy,
Finance, Auto, Retail): ").strip()
    start_date_input = input("Enter the start date for historical
data (YYYY-MM-DD): ").strip()
    end_date_input = input("Enter the end date for historical data
(YYYY-MM-DD): ").strip()
    forecast_horizon = int(input("Enter the number of future
business days to forecast: ").strip())
    print("\n\n")

    # Download historical data
    new_data = yf.download(user_ticker, start=start_date_input,
end=end_date_input)
    if new_data.empty:
        print("No data found for this ticker in the given date
range.")
        return
    new_data = new_data[['Close']].reset_index()

    # Create DataFrame with metadata
    new_df = new_data.copy()
    new_df["Ticker"] = user_ticker
    new_df["Sector"] = user_sector
    new_df = new_df.sort_values("Date").reset_index(drop=True)

    # Scale the data
    scaler_new = MinMaxScaler(feature_range=(0, 1))
    new_df["Scaled"] = scaler_new.fit_transform(new_df[["Close"]])

    # For a new ticker, assign StockID = old_n_stocks (since 0 to
old_n_stocks-1 are taken)
    new_stock_id = old_n_stocks
    known_sectors = ["Tech", "Energy", "Finance", "Auto", "Retail"]
    if user_sector in known_sectors:
        new_sector_id = known_sectors.index(user_sector)
    else:
        new_sector_id = 0
        print("Warning: Unrecognized sector; defaulting to sector ID
0.")

    new_df["StockID"] = new_stock_id
    new_df["SectorID"] = new_sector_id

    # Create sequences from the historical data
    X_new_seq, y_new, _, _ = create_universal_sequences(new_df,
time_steps)
    X_new_seq = X_new_seq.reshape(X_new_seq.shape[0], time_steps, 1)

    # Prepare constant inputs for the new ticker
    num_samples = X_new_seq.shape[0]
    X_new_stock = np.full((num_samples, 1), new_stock_id)
    X_new_sector = np.full((num_samples, 1), new_sector_id)
```

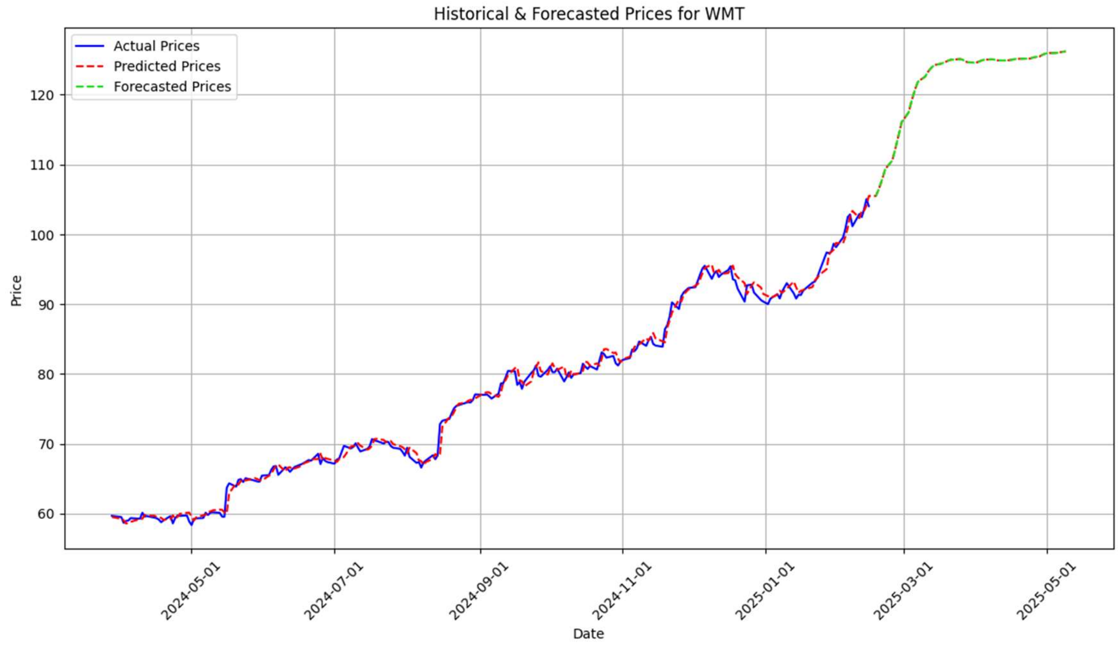
```
    # Expand the pre-trained model to include the new ticker
    new_model = expand_model_for_new_ticker(old_model, old_n_stocks,
user_ticker)

    # Fine-tune the new model on the historical data
    fine_tune_lr = 1e-4
    new_model.compile(optimizer=Adam(learning_rate=fine_tune_lr,
clipvalue=1.0), loss='mse')
    fine_tune_callbacks = [EarlyStopping(monitor='loss', patience=3,
restore_best_weights=True, verbose=1)]

    new_model.fit(
        [X_new_seq, X_new_stock, X_new_sector], y_new,
        epochs=20,
        batch_size=32,
        callbacks=fine_tune_callbacks,
        verbose=1
    )

    # Evaluate on historical data
    preds_new_scaled = new_model.predict([X_new_seq, X_new_stock,
X_new_sector])
    preds_new = scaler_new.inverse_transform(preds_new_scaled)
    y_new_actual = scaler_new.inverse_transform(y_new.reshape(-1,
1))

    rmse_new = math.sqrt(mean_squared_error(y_new_actual,
preds_new))
    print("------------------------------------------------------------
--------------------")
    print(f"Fine-Tuned Model RMSE on {user_ticker} Data:
{rmse_new:.4f}")
    print("------------------------------------------------------------
--------------------")
    print("\n\n")

    # Historical dates corresponding to sequence targets
    historical_dates =
new_df["Date"].iloc[time_steps:].reset_index(drop=True)

    # Iterative forecasting: use the last sequence as seed
    last_sequence = X_new_seq[-1]  # shape: (time_steps, 1)
    stock_input_forecast = np.array([[new_stock_id]])
    sector_input_forecast = np.array([[new_sector_id]])
    forecast_scaled = iterative_forecast(new_model, last_sequence,
stock_input_forecast, sector_input_forecast, forecast_horizon)
    forecast_scaled = np.array(forecast_scaled).reshape(-1, 1)
    forecast_prices = scaler_new.inverse_transform(forecast_scaled)

    # Generate future dates (business days) starting the day after
the last historical date
    last_date = new_df["Date"].max()
```

```
    future_dates = pd.date_range(start=last_date +
pd.Timedelta(days=1), periods=forecast_horizon, freq='B')

    # Combine historical and forecast data for a single plot
    combined_dates = pd.concat([historical_dates,
pd.Series(future_dates)], ignore_index=True)
    # Ensure shapes match: create NaN array with shape
(forecast_horizon, 1)
    nan_array = np.full((forecast_horizon, 1), np.nan)
    combined_actual = np.concatenate([y_new_actual, nan_array],
axis=0)
    combined_predicted = np.concatenate([preds_new,
forecast_prices], axis=0)

    plt.figure(figsize=(14, 7))
    plt.plot(combined_dates, combined_actual, label="Actual Prices",
color="blue")
    plt.plot(combined_dates, combined_predicted, label="Predicted
Prices", color="red", linestyle="--")
    # Highlight forecasted portion in green
    plt.plot(future_dates, forecast_prices, label="Forecasted
Prices", color="#22DD22", linestyle="--")

    plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-
%d'))
    plt.gca().xaxis.set_major_locator(mdates.AutoDateLocator())
    plt.xticks(rotation=45)
    plt.title(f"Historical & Forecasted Prices for {user_ticker}")
    plt.xlabel("Date")
    plt.ylabel("Price")
    plt.legend()
    plt.grid(True)
    plt.show()

# --------------------------------------------------------------------
-----------
# Call the function for user input and display forecast
# --------------------------------------------------------------------
-----------
predict_and_forecast_new_stock()
```

```
Enter the stock ticker for prediction (e.g., IBM):  WMT
Enter the stock sector (e.g., Tech, Energy, Finance, Auto,
Retail):  Retail
Enter the start date for historical data (YYYY-MM-DD):  2024-
01-01
Enter the end date for historical data (YYYY-MM-DD):  2025-02-
18
Enter the number of future business days to forecast:  60
```

```
[*********************100%***********************]  1 of 1 completed



Epoch 1/20


7/7 ━━━━━━━━━━━━━━━━━━━━ 3s 24ms/step - loss: 5.7034e-04
Epoch 2/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step - loss: 5.5962e-04
Epoch 3/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 15ms/step - loss: 5.4266e-04
Epoch 4/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step - loss: 5.2899e-04
Epoch 5/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 4.0796e-04
Epoch 6/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 4.5966e-04
Epoch 7/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 5.4144e-04
Epoch 8/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 0s 13ms/step - loss: 5.0874e-04
Epoch 8: early stopping
Restoring model weights from the end of the best epoch: 5.
7/7 ━━━━━━━━━━━━━━━━━━━━ 1s 60ms/step
--------------------------------------------------------------------------
---
Fine-Tuned Model RMSE on WMT Data: 0.8973
--------------------------------------------------------------------------
---



1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 24ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
```



Conclusion

This study has demonstrated the effectiveness of deep learning techniques in the challenging task of stock price prediction. By leveraging historical data from Yahoo Finance and employing advanced architectures—namely LSTM and Bidirectional LSTM—the project showcased a robust framework capable of capturing both short-term fluctuations and long-term trends. The universal model design, which integrates sequential price data with metadata embeddings (for stock and sector identifiers), allowed the models to learn market-specific nuances effectively. Notably, the Bi-LSTM model consistently achieved lower error metrics compared to the standard LSTM, highlighting its advantage in processing bidirectional temporal dependencies. Moreover, the ensemble approach—combining predictions from both models—further enhanced the overall predictive accuracy. The framework's ability to expand and fine-tune for new tickers underscores its scalability and adaptability, making it a promising tool for dynamic financial forecasting.

Future Work

While the current methodology lays a solid foundation for stock price prediction using deep learning, several avenues can be explored to further enhance its performance and applicability:

- **Incorporation of Additional Data Sources:**
    - **Sentiment Analysis:** Integrate qualitative data such as financial news, social media sentiment, and expert opinions to complement historical price data. This could provide a more holistic view of market conditions.
    - **Macroeconomic Indicators:** Include economic variables (e.g., interest rates, GDP growth, inflation) to capture broader market influences that affect stock prices.
- **Model Enhancements and Hybrid Approaches:**
    - **Ensemble and Hybrid Models:** Investigate the potential of combining traditional statistical models (e.g., ARIMA, GARCH) with deep learning architectures to balance interpretability and prediction accuracy.
    - **Feature Engineering:** Develop more sophisticated feature extraction techniques, such as technical indicators or wavelet transforms, to enhance the input data quality.
- **Real-Time Forecasting and Adaptive Learning:**
    - **Online Learning:** Adapt the models for real-time data streams, enabling continuous learning and immediate adjustments to changing market conditions.
    - **Adaptive Model Updating:** Explore mechanisms for periodic retraining or fine-tuning to ensure that the models remain robust against evolving market dynamics.
- **Scalability and Deployment:**
    - **Broader Market Application:** Extend the framework to cover additional markets and asset classes, assessing its performance across diverse economic environments.
    - **Deployment in Production:** Develop a scalable, user-friendly interface for real-time stock prediction and portfolio management, which could be valuable for both retail and institutional investors.
