Analytics of Stock Pricing Using Machine Learning Techniques

32513 31005 Advanced Data Analytics Algorithms, Machine Learning – Spring 2024

24629661

HYUNMIN KIM

UTS

Table of Contents

Background

Due to the rising volatility and complexity in financial markets, accurate stock price prediction has become increasingly essential. However, complex needs need to be made more accessible for investors and financial analysts. Since stock price movements are driven by a mixture of economic, political and social factors that keep evolving with time, traditional statistical models need to be more robust to capture the dynamic patterns. The rise of machine learning. Due to recent advances in data science and the increased accessibility of enormous financial datasets, tackling this issue has become feasible via powerful machine-learning techniques. What causes stock pricing analytics via machine learning to be completed is the ability of these modern algorithms to extract further information about share prices with more precise predictions. However, with the capacity of machine learning models to understand past data and trace obscure patterns found in financial markets, these can be a boon for enhancing prediction accuracies that assist investors alike. In this study, we plan to investigate various machine learning methods in analyzing historical stock data and examine the accuracy of performance prediction from each method with the goal of identifying what kind yields more accurate insights for stock market forecasts.

Problem Statement

Object: Design a machine learning model to predict a currency pair's future opening and closing price, AUD/USD, based on the historical time series style dataset. This includes the Model identifying long-standing trends in the past data using differing network architectures

and data over the long period.

Input: This report consists of historical financial data and after-preprocessing data. In particular, the historical data is the AUD/USD pair's Open, Close, high, low prices of the past 15 years, which is obtained from Yahoo Finance using the yfinance library. This data can help to model the future price of the currency pair based on its past trends. To ensure that the Model is consistent and to increase the model performance, preprocessing are applied on the data, such as using MinMaxScaler or StandardScaler to normalize the data ranges throughout the time frames. In addition, a log transformation is used to stabilize the variance and work on skewness which helps to make the time-series authentic and stabilize. Correspondingly, this data is split into the train and test data. This ensures that the Model learns on the train data and tests and validates it the unseen test data.

Output: The output of this report is the various performance metrics and visualization on Model so that LSTM, LSTM + GRU, and LSTM + GCN. For instance, it includes the Mean Squared Error (MSE), Mean Absolute Error (MAE), and R^2 score with six precision decimal to see how the Model has performed in predicting the Open and Close prices of the currency pair and predictions between Open and Close prices, and the residual plot. It is essential because it helps visualize the predicted differences and actual values. The true price and predicted points of Open and Close prices are also plotted. Other plotting outputs, such as error distributions and predictions, and comparison datasets can help determine how effectively the Model can predict the future data by capturing the trend and exceptional anomalies on the financial time-series dataset.

Data Preprocessing

```python
# 1. Data collection and preprocessing
import pandas as pd
import numpy as np
import yfinance as yf
from sklearn.preprocessing import MinMaxScaler

# Fetch AUD/USD data from Yahoo Finance
ticker = 'AUDUSD=X'
data = yf.download(ticker, interval='1d', start=(pd.Timestamp.now() - pd.DateOffset(years=15)).strftime('%Y-%m-%d'), end=pd.Timestamp.now().strftime('%Y-%m-%d'))

# Select necessary features (e.g., Open, High, Low, Close)
features = ['Open', 'High', 'Low', 'Close']
data = data[features]

# Apply log transformation for scaling the data
data = data.apply(lambda x: np.log1p(x))

# Normalize the data
scaler = MinMaxScaler()
data_scaled = scaler.fit_transform(data)

# Create a DataFrame with the normalized data
data_scaled_df = pd.DataFrame(data_scaled, columns=features)

# Print the first 100 rows of the data
print(data_scaled_df.head(100))

# Print the total number of data points
print(f'Total number of data points: {len(data_scaled_df)}')
```

```
[*********************100%***********************]  1 of 1 completed
0   0.661236  0.650091  0.678235  0.660454
1   0.676084  0.666067  0.694720  0.677540
2   0.686707  0.681502  0.704301  0.692294
3   0.678352  0.666527  0.696890  0.678598
4   0.698822  0.690083  0.716033  0.700132
..       ...       ...       ...       ...
95  0.649827  0.649789  0.648246  0.635759
96  0.635916  0.627962  0.643974  0.638669
97  0.639413  0.627518  0.634257  0.624341
98  0.624772  0.632411  0.645625  0.641441
99  0.646883  0.639568  0.657820  0.649800

[100 rows x 4 columns]
Total number of data points: 3907
```

Preprocessing the data is a foundation stone for constructing machine learning models for financial time series prediction to guarantee the accuracy and robustness of our model. The primary financial dataset used in the report includes 15 years of historical data for the AUD/USD currency pair with Open, High, Low and Close prices. Preprocessing is essential to preprocess the raw data into machine learning algorithms as it will convert all complex and volatile financial market information. This section will cover the main preprocessing steps taken on the dataset before model fit and prediction.

```
# Fetch data from Yahoo Finance
ticker = "AUDUSD=X"
start_date = pd.Timestamp.now() - pd.DateOffset(years=15)
end_date = pd.Timestamp.now()

data = yf.download(ticker, start=start_date, end=end_date)

# Keep only "Open" and "Close" columns
data = data[['Open', 'Close']]
```

The first step consisted of obtaining the data through finance, which returned a dataset with all prices for that currency pair at a daily frequency. The data was then cleaned from missing values and inconsistencies in order to avoid biases that would be introduced within the algorithms due to non-standard related (columnValues) along record attributes, but also logically, it makes sense, let us say, for example, situations in which your algorithm removes or fills these null Values with a fixed value we might start using another NotebookClass—data version. Two normalization techniques were used to ensure uniform scale across the different price scales: StandardsScaller and Min Max Scaler. Scaling techniques are used to help the model interpret data more effectively by normalizing feature ranges, which avoids high bias towards larger values such as High or Close prices. When we wanted a more Gaussian-like distribution, we used StandardScaler to standardize the features by removing the mean and scaling them to unit variance. MinMaxScaler: This scaler is proper when we must keep the proportional relationship between maximum and minimum values in an arbitrary but fixed range from (0-1).

```
# Preprocessing
scaler = MinMaxScaler(feature_range=(0, 1))
data_scaled = scaler.fit_transform(data)
```

Normalization was also coupled with a logarithmic transformation to reduce variance and skewness in the data. It is significant in time series forecasting because financial data often has exponential increases or decreases, which causes heteroscedasticity, which means our variance of residuals differs over any period by applying np. Here, log1p(), the model will be able to handle those changes in variance better and not overfit as much.

```
# Preprocessing
scaler = MinMaxScaler(feature_range=(0, 1))
data_scaled = scaler.fit_transform(data)

# Prepare datasets for LSTM (predict both Open and Close prices)
def create_lstm_dataset(data, time_step):
    X, Y = [], []
    for i in range(len(data) - time_step - 1):
        X.append(data[i:(i + time_step), :])
        Y.append(data[i + time_step, :])  # Predicting both 'Open' and 'Close'
    return np.array(X), np.array(Y)

X, Y = create_lstm_dataset(data_scaled, time_step)

# Train-test split (80% training, 20% testing)
train_size = int(len(X) * 0.8)
X_train, X_test = X[:train_size], X[train_size:]
Y_train, Y_test = Y[:train_size], Y[train_size:]
```

From this dataset, training and testing sets have been split with a standard 80–20 ratio to ensure that the model has seen enough data for learning but is still evaluated on mostly unseen new data. To address the above issues, time step windows were introduced, forming sequences of historical data that could be learned from a model. With a window size of 30 days, the moving average was calculated as follows to get an idea of both short and medium-term trends in how the currency pair moved. It, therefore, creates a model for each 30-day sequence and predicts the Open of 31st-day price based on input (specifically close prices) similar to the buy scenario.

Finally, data integrity was ensured by maintaining the consistency of input features (Open and Close prices) across all models tested, i.e., Long Short-Term Memory (LSTM), LSTM in conjunction with Gate Recurrent Units (GRU) and LSTM along with Graph Convolutional Networks(GCN). These preprocessing steps aimed to give models a standardized, cleaned, well-structured dataset to make consistent comparisons and impact performance metrics.

Machine Learning Classification

LSTM

Given that they deal with data through time and are good at preserving both short-term dependences and long-term dependencies, the LSTM models can be highly efficient for stock predicting prices. Past price movements often affect future trends in financial markets, and LSTM utilizes a memory cell structure to store valuable information for proper decisions while eliminating redundancy. Item three: LSTM can be used irregularly in abnormal patterns (for example, when prices are very volatile and there is a slow trend shifting). Concurrent with multiple variables(multiple speeds) to predict open price + close Priceinity to improve accuracy. These are the characteristics that make the LSTM model powerful and best suited when it comes to handling financial market data.**(**Phuoc et al., 2024)

```
import numpy as np
import pandas as pd
import yfinance as yf
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
import matplotlib.pyplot as plt

# Parameters
epochs = 50  # Same number of epochs as in the LSTM + GCN model
batch_size = 16
time_step = 30  # Same time step as in LSTM + GCN

# Fetch data from Yahoo Finance
ticker = "AUDUSD=X"
start_date = pd.Timestamp.now() - pd.DateOffset(years=15)
end_date = pd.Timestamp.now()

data = yf.download(ticker, start=start_date, end=end_date)

# Keep only "Open" and "Close" columns
data = data[['Open', 'Close']]

# Preprocessing
scaler = MinMaxScaler(feature_range=(0, 1))
data_scaled = scaler.fit_transform(data)

# Prepare datasets for LSTM (predict both Open and Close prices)
def create_lstm_dataset(data, time_step):
    X, Y = [], []
    for i in range(len(data) - time_step - 1):
        X.append(data[i:(i + time_step), :])
        Y.append(data[i + time_step, :])  # Predicting both 'Open' and 'Close'
    return np.array(X), np.array(Y)

X, Y = create_lstm_dataset(data_scaled, time_step)

# Train-test split (80% training, 20% testing)
train_size = int(len(X) * 0.8)
X_train, X_test = X[:train_size], X[train_size:]
Y_train, Y_test = Y[:train_size], Y[train_size:]

# Build LSTM model
lstm_model = Sequential()
lstm_model.add(LSTM(100, return_sequences=True, input_shape=(time_step, X.shape[2])))
lstm_model.add(Dropout(0.3))
lstm_model.add(LSTM(100, return_sequences=False))
```

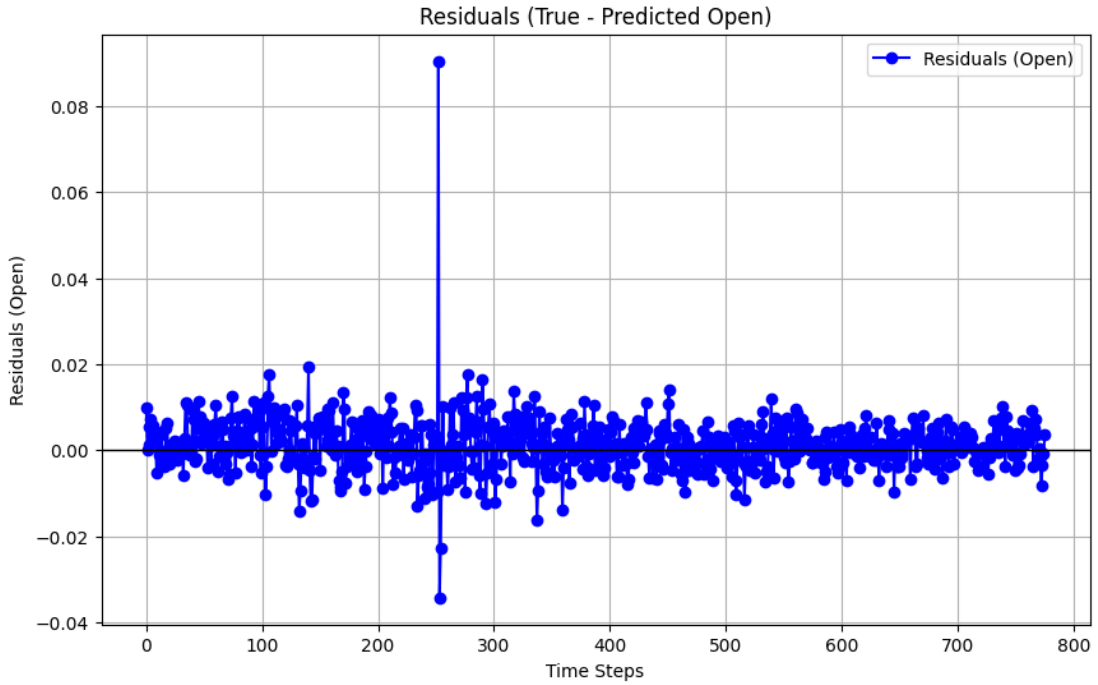
```
# Model training
lstm_model.fit(X_train, Y_train, epochs=epochs, batch_size=batch_size, validation_data=(X_test, Y_test))

# Predictions
train_pred = lstm_model.predict(X_train)
test_pred = lstm_model.predict(X_test)

# Rescale predictions
train_pred_rescaled = scaler.inverse_transform(train_pred)
test_pred_rescaled = scaler.inverse_transform(test_pred)
Y_train_rescaled = scaler.inverse_transform(Y_train)
Y_test_rescaled = scaler.inverse_transform(Y_test)
```

For this analysis, an LSTM model was trained to predict both the open and close prices. To prevent overfitting, the model architecture consisted of two LSTM layers with 100 units in each Dropout layer. A dense output layer with two neurons, one for predicting the open price and one for predicting the close price. The model was trained for 50 epochs with a batch size of 16, and early stopping was applied to prevent overfitting during training.

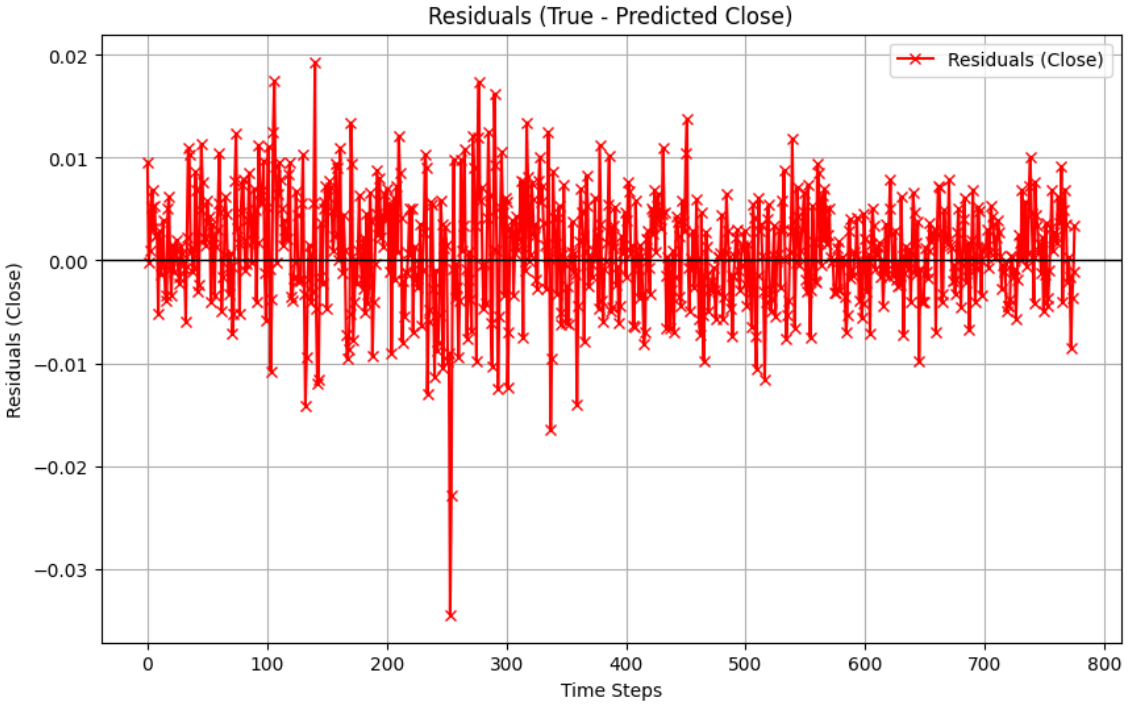
```
True vs Predicted values (first 5 samples):
Open Prices:
   True Open  Predicted Open
0   0.751690        0.741808
1   0.746531        0.746386
2   0.747000        0.745811
3   0.749450        0.743883
4   0.751430        0.744294

Close Prices:
   True Close  Predicted Close
0    0.751530         0.741941
1    0.746370         0.746516
2    0.746910         0.745940
3    0.749180         0.744013
4    0.751298         0.744424

Performance metrics for Open:
LSTM Open MSE: 0.000040
LSTM Open MAE: 0.004225
LSTM Open R² Score: 0.954468

Performance metrics for Close:
LSTM Close MSE: 0.000029
LSTM Close MAE: 0.004083
LSTM Close R² Score: 0.966792
```

The tables show the first 5 predicted and true values for both open and close prices.

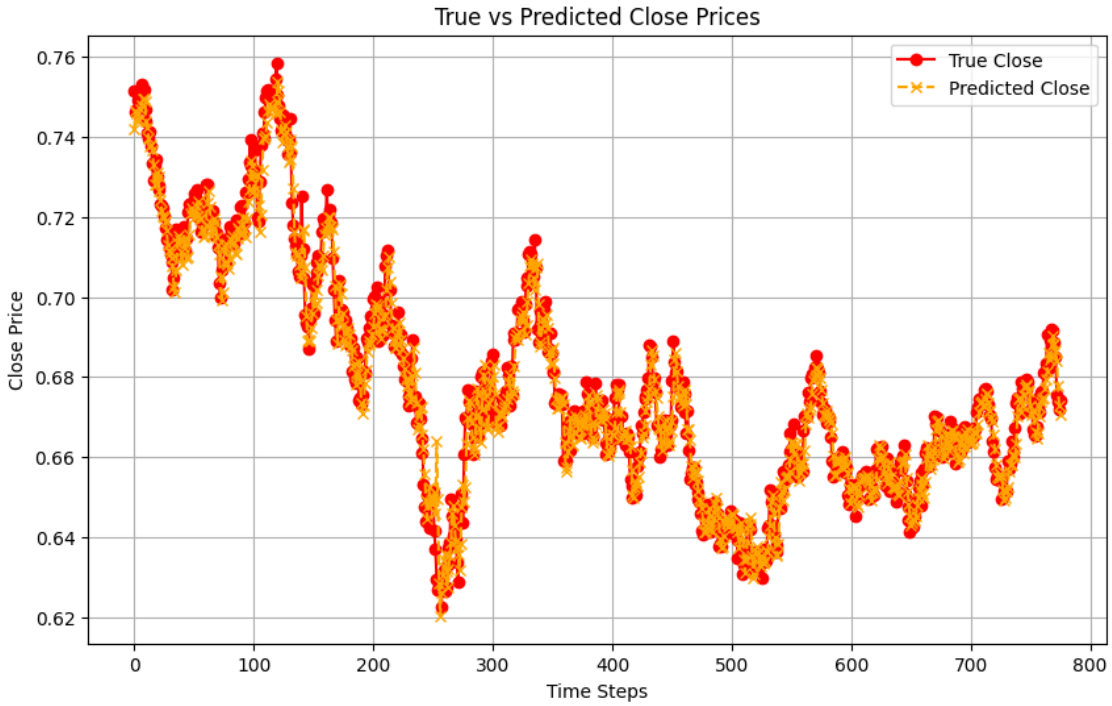
The performance of the model was assessed based on open and close price features by the following metrics: Mean Squared Error (MSE): Measures the average squared difference between predicted values. Sum of Absolute Difference: The average absolute difference between predicted (forecasting) values and actual data is calculated. R^2 Score Out r2 = 0.897 Indicates the percentage of variance in the dependent variable that can be explained by changes to the independent variables, or R squared value None the less, we can see from these metrics that with this model, open and close prices could be predicted very well or at ease especially for close where R^2=0.966792



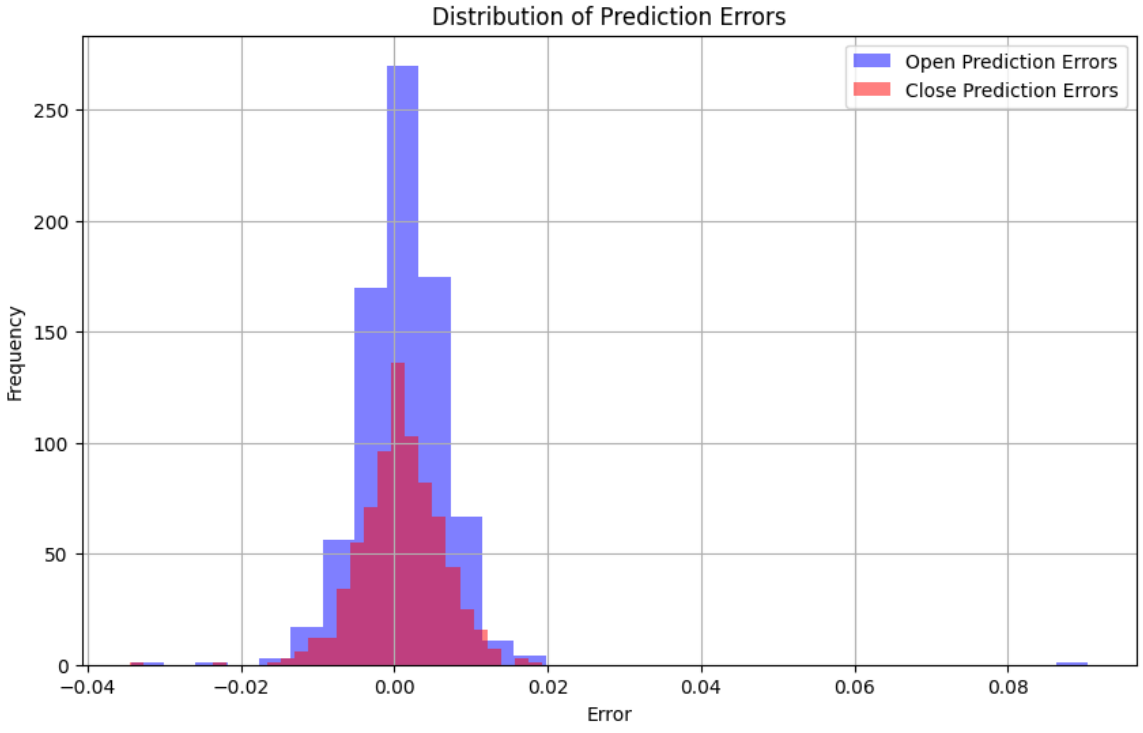
Open Prices Residual Plot In the plot, it appears that most of the residuals are clustered near zero with a few stand alone data points. This means that the model is right most of the time in predicting opening prices, but there are some places on the timeline where it fails to predict accurately.



The residuals for the close prices are also centered at zero but are slightly more spread out than we saw in open. This means that even though the model is doing well overall, there are certain times when its predictions need to catch up.



The next graph represents the actual and forecasted close prices during the test period, indicating that our model captures trends in close prices rather well with some divergence even on peak points.




```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
Predicted Open price for next day: 0.671080
Predicted Close price for next day: 0.671632
```

Plotted a histogram to see how the errors are distributed for open and closed prices. A small spread of prediction errors means that the model makes many precise predictions with few outstanding large error values.

The predicted open and close prices are displayed at the end for the next day based on the most recent data in the test set after training the LSTM model.

I can conclude that the LSTM algorithm did an excellent job predicting open and close prices for AUD/USD with high R^2 scores at low MSE values. Although the model does well, residuals and performance metrics have shown some cases where predictions tend to be very far from fundamental values (incredibly open prices). In general, we believe that LSTM models offer characteristics beneficial for forecasting time-series data, and as such, the results of this analysis support their ability to be utilized in financial forecasting.

LSTM + GRU

Combining LSTM with GRU provides a better machine learning option than using only the LSTM model because both have advantages. Through this combination, we are gaining them all for an effective prediction. While LSTM is excellent in modeling long-term dependencies, it can also be computationally expensive due to its complexity (thanks to the added complexity from multiple gates). By contrast, GRU is a simpler version that consolidates the forget and input gates to the update gate to ease the computational burden and fast train but maintain key information. (Fouzan, 2023)

LSTM is better than GRU in memory retention. On the other hand, a gated Recurrent Unit (GRU) is a faster and simpler RNN cell, which you can implement to work for small, shorter pattern sequence data where LSTM gives the worse performance. So, using both might be beneficial as it allows us to take advantage of the properties of each one we want while sacrificing the main drawback by keeping the LSTMwhile picking the best part from different cell types. Such a hybrid approach can provide both accuracy and efficiency. It will be particularly relevant in forecasting financial time series, where the data consists of long-term trends with short-term cycles. This can also help to solve another problem: overfitting. It could also significantly improve training speeds for specific problems since the combined networks would not have a single architecture but two of them.

```python
import numpy as np
import pandas as pd
import yfinance as yf
import tensorflow as tf

from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from tensorflow.keras import models, layers
import matplotlib.pyplot as plt

# Parameters
epochs = 50
batch_size = 16
time_step = 30

# Fetch data from Yahoo Finance
ticker = "AUDUSD=X"
start_date = pd.Timestamp.now() - pd.DateOffset(years=15)
end_date = pd.Timestamp.now()

data = yf.download(ticker, start=start_date, end=end_date)

# Keep only "Open" and "Close" columns
data = data[['Open', 'Close']]

# Preprocessing
scaler = MinMaxScaler(feature_range=(0, 1))
data_scaled = scaler.fit_transform(data)

# Prepare datasets for LSTM + GRU
def create_lstm_gru_dataset(data, time_step):
    X, Y_open, Y_close = [], [], []
    for i in range(len(data) - time_step - 1):
        X.append(data[i:(i + time_step), :])
        Y_open.append(data[i + time_step, 0])  # Open price
        Y_close.append(data[i + time_step, 1])  # Close price
    return np.array(X), np.array(Y_open), np.array(Y_close)

X, Y_open, Y_close = create_lstm_gru_dataset(data_scaled, time_step)

# Train-test split (80% training, 20% testing)
train_size = int(len(X) * 0.8)
X_train, X_test = X[:train_size], X[train_size:]
Y_train_open, Y_test_open = Y_open[:train_size], Y_open[train_size:]
Y_train_close, Y_test_close = Y_close[:train_size], Y_close[train_size:]

# Build LSTM + GRU model
gru_lstm_model = models.Sequential()
gru_lstm_model.add(layers.LSTM(100, return_sequences=True, input_shape=(time_step, X.shape[2])))
gru_lstm_model.add(layers.GRU(100, return_sequences=False))
gru_lstm_model.add(layers.Dense(2))  # Predicting both 'Open' and 'Close'
```

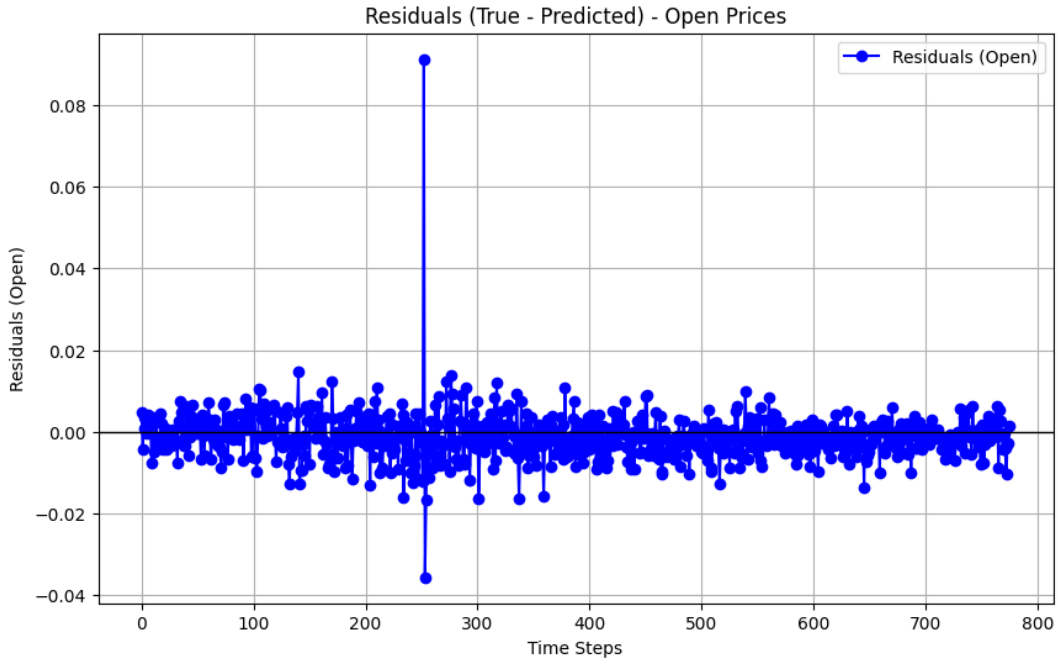
```python
# Model training
gru_lstm_model.fit(X_train, np.column_stack((Y_train_open, Y_train_close)), epochs=epochs, batch_size=batch_size, validation_data=(X_test, np.column_stack((Y_test_open, Y_test_close))))

# Predictions
train_pred = gru_lstm_model.predict(X_train)
test_pred = gru_lstm_model.predict(X_test)

# Rescale predictions
train_pred_rescaled = scaler.inverse_transform(np.column_stack((train_pred[:, 0], train_pred[:, 1])))
test_pred_rescaled = scaler.inverse_transform(np.column_stack((test_pred[:, 0], test_pred[:, 1])))
Y_train_rescaled = scaler.inverse_transform(np.column_stack((Y_train_open, Y_train_close)))
Y_test_rescaled = scaler.inverse_transform(np.column_stack((Y_test_open, Y_test_close)))
```

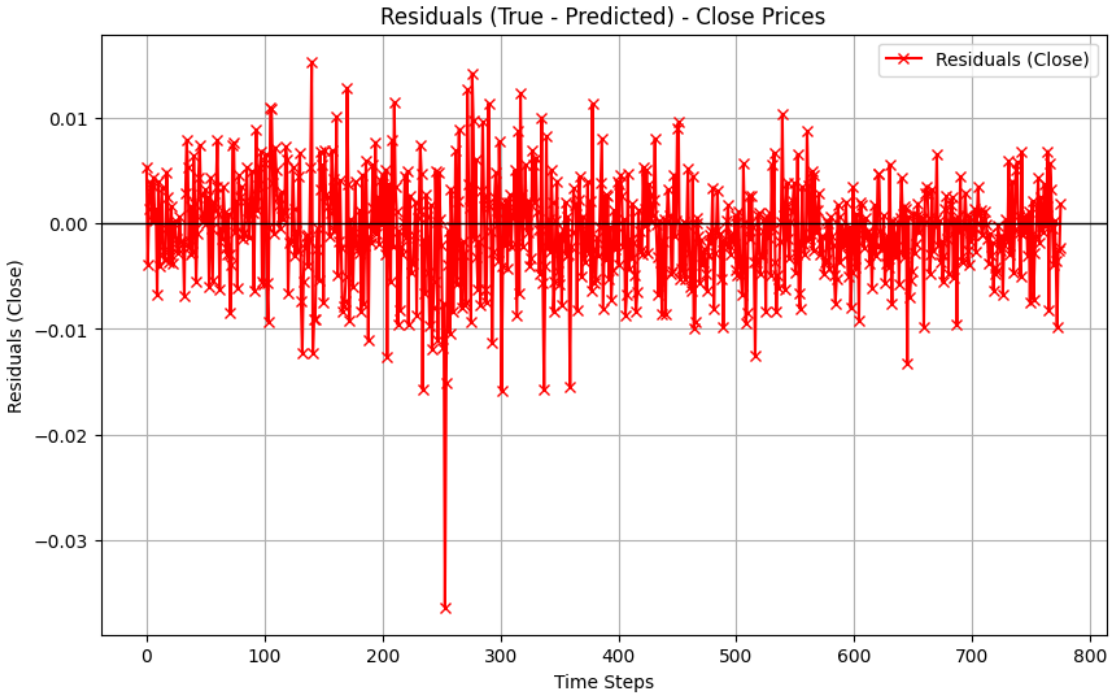
LSTM+GRU stacked two recurrent layers. Long short-term memory (100 units): this layer is responsible for retaining long-term dependencies in the data. The return sequences=True argument simply guarantees that the output of the LSTM layer will be passed as input to the next GRU Layer. The output from the LSTM (100 units) layer is fed to GRU (100 units), which becomes more specialized in predicting the next step in the sequence. The last Dense layer is also the output; this one output two values only because it also predicts the Open and Close prices of a time. Such dual layers help the model combine both grid and LSTM

strengths through a process of improvement in predictions.

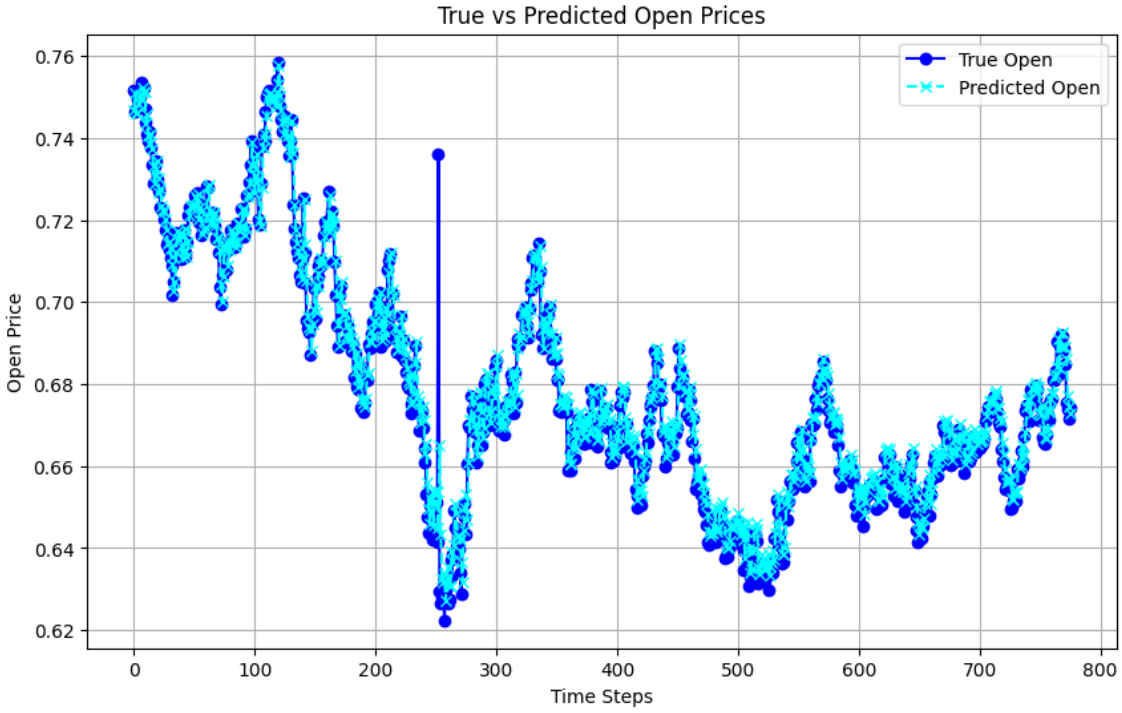
```
True vs Predicted values (first 5 samples):
   True Open  Predicted Open  True Close  Predicted Close
0   0.751690        0.746894    0.751530         0.746179
1   0.746531        0.751002    0.746370         0.750306
2   0.747000        0.746166    0.746910         0.745532
3   0.749450        0.746029    0.749180         0.745310
4   0.751430        0.748454    0.751298         0.747705
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step

Predicted Open for the next day: 0.676485
Predicted Close for the next day: 0.676001

LSTM + GRU Open MSE: 0.000036
LSTM + GRU Open MAE: 0.003864
LSTM + GRU Open R² Score: 0.959158
LSTM + GRU Close MSE: 0.000024
LSTM + GRU Close MAE: 0.003665
LSTM + GRU Close R² Score: 0.972415
```

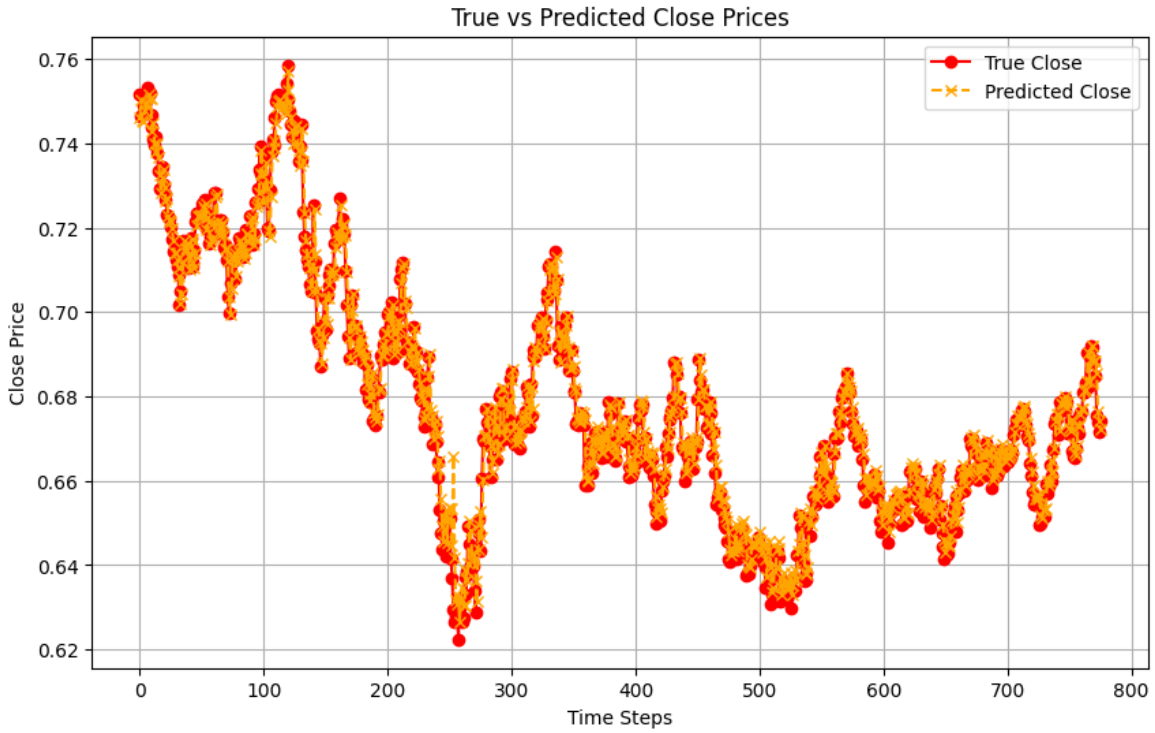
Performance of the LSTM + GRU model evaluation — Open and Close prices the above metrics prove that it is highly accurate and can predict close prices better than other indicators (R^2 score: 0.972). Additionally, the shallow values of MSE and MAE confirm that the model can reduce error significantly.



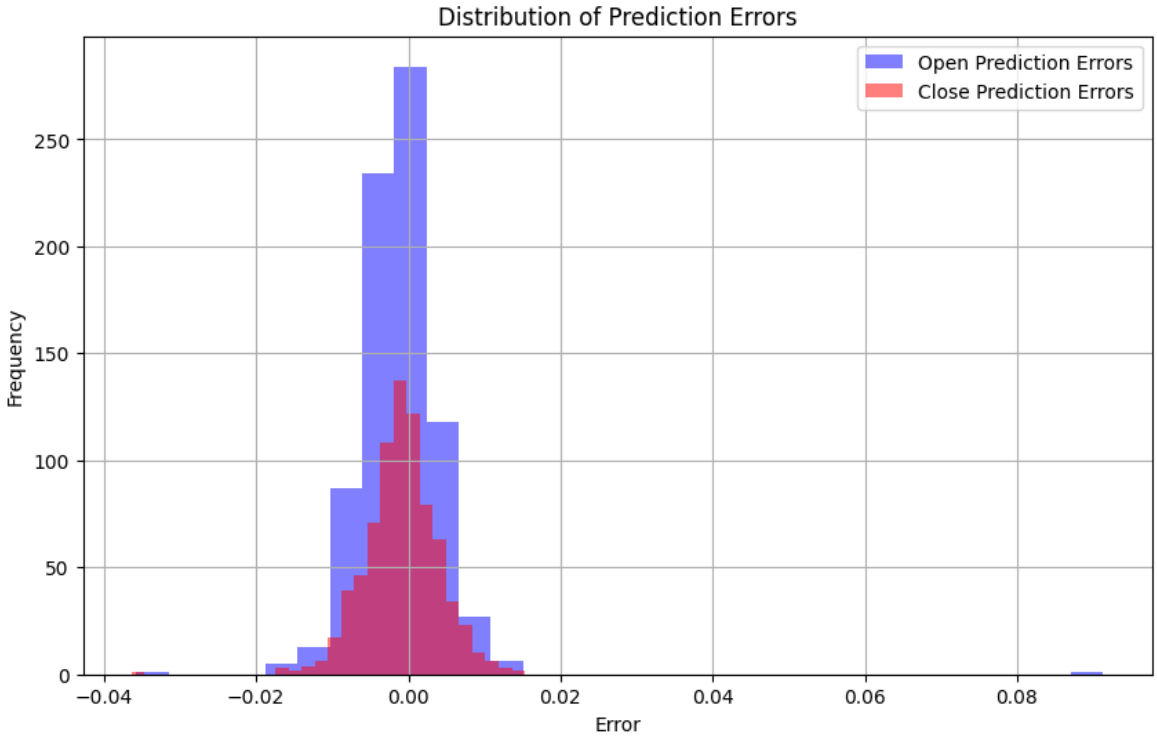


Residuals Plot (Price Open vs. Price Close): The residuals plot, in combination with the predicted values, helps us to visualize where our model price predictions differed positively, such as a positive residual means that we under-predicted and vise-versa for negative cases since it compares Real prices against Predictions, so linearity straight lien or most minor difference. For Open and Close, the residuals of most predictions are close to zero, which makes sense as this is precisely what we predicted. We observe some outliers, indicating the model performance weakness to substantial price changes — Open prices plot.





True vs. Predicted Prices Plot: Apart from some minor variations our model has captured the general trends in both Open and Close prices with equal precision (at least at this point). The overlapping of the plotted lines indicates that the model has done an excellent job of capturing how prices change over time.



Prediction errors: The distribution of prediction errors in the Open and Close prices indicates that most are not far from zero. It is another proof that the case model can predict well and deviate very little from true values.

UTS

LSTM and GRU layers combined have shown good predictive power in predicting Open and Close and in forecasting AUD/USD prices. The model's R^2 scores and MSE/MAE values suggest that it can capture these temporal dependencies well in financial data. The combination of LSTM and gate control strategies provides a powerful alternative to using only an LSTM for time-series forecasting in the financial domain, which also presents the idea that combining various types of recurrent architectures may help improve prediction accuracy with such a hybrid approach. Refining and performance are optimizing this model. Further, it could be used in real-world trading by making assistant buy-sell predictions based on predicted price movements.

LSTM + GCN

Since the LSTM + GCN model can combine well the long-term dependency modeling of Long Short-Term Memory (LSTM) and graph-structured local neighborhood aggregation computation from Graph Convolutional Networks (GCNs), we obtain a better R^2 value based on this homogeneously initial preprocessing. Long Short-Term Memory (LSTM) networks are very effective in time series forecasting because they can remember patterns and trends that occur across stock price sequences at a given interval. This allows the model to know how time points from the past influence future price predictions. Specifically, GCNs are good at learning relationships between connected nodes corresponding to different time steps in the data. GCNs on the output of LSTM not only exploit temporal patterns [56, 61] but also model dependency between consecutive time steps. Such a hybrid approach helps the model represent these complexities in financial time series data with an improved generalization and variance-capturing capability, as seen when using the R^2 score for our accuracy metric. This synergy of temporal capturing LSTM and relational learning GCN leads to improved predictive power for event detection. (Azati et al., 2024)

```
# Importing libraries
import pandas as pd
import numpy as np
import torch
import torch.nn.functional as F
import yfinance as yf
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
from torch_geometric.nn import GCNConv
from torch_geometric.data import Data
import matplotlib.pyplot as plt

# 1. Data collection and preprocessing
ticker = 'AUDUSD=X'
data = yf.download(ticker, interval='1d', start=(pd.Timestamp.now() - pd.DateOffset(years=15)).strftime('%Y-%m-%d'), end=pd.Timestamp.now().strftime('%Y-%m-%d'))
features = ['Open', 'Close']  # Select only 'Open' and 'Close' values
data = data[features]

# Scaling the data using standardization
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data)

# Function to create the dataset for LSTM
def create_lstm_dataset(data, time_step=1):
    X, Y = [], []
    for i in range(len(data) - time_step - 1):
        X.append(data[i:(i + time_step), :])  # Use 'Open' and 'Close' as features
        Y.append(data[i + time_step])  # Predict 'Open' and 'Close'
    return np.array(X), np.array(Y)

time_step = 30  # Set time step to 30 days
# Creating the LSTM dataset
X_price, Y_price = create_lstm_dataset(data_scaled, time_step)

# Splitting the dataset into training and test sets
X_train_price, X_test_price, Y_train_price, Y_test_price = train_test_split(X_price, Y_price, test_size=0.2, random_state=42)

# 2. Define and train the LSTM model
lstm_model_price = Sequential()
lstm_model_price.add(LSTM(100, return_sequences=True, input_shape=(time_step, X_price.shape[2])))
lstm_model_price.add(Dropout(0.3))  # Add dropout to prevent overfitting
lstm_model_price.add(LSTM(100, return_sequences=False))
lstm_model_price.add(Dropout(0.3))

# Output two values to predict 'Open' and 'Close'
lstm_model_price.add(Dense(2))  # Extract features to pass to GCN
lstm_model_price.compile(optimizer='adam', loss='mean_squared_error')
```

```
# Set early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)

# Train the LSTM model
lstm_model_price.fit(X_train_price, Y_train_price, epochs=50, batch_size=16, validation_data=(X_test_price, Y_test_price), callbacks=[early_stopping])

# LSTM output
lstm_train_output = lstm_model_price.predict(X_train_price)
lstm_test_output = lstm_model_price.predict(X_test_price)

# 3. Prepare data for GCN
num_nodes_train = lstm_train_output.shape[0]
num_nodes_test = lstm_test_output.shape[0]

# Create adjacency matrix
edges_train = torch.tensor([[i, i + 1] for i in range(num_nodes_train - 1)], dtype=torch.long).t().contiguous()
edges_test = torch.tensor([[i, i + 1] for i in range(num_nodes_test - 1)], dtype=torch.long).t().contiguous()

x_train = torch.tensor(lstm_train_output, dtype=torch.float)
x_test = torch.tensor(lstm_test_output, dtype=torch.float)

gcn_train_data = Data(x=x_train, edge_index=edges_train)
gcn_test_data = Data(x=x_test, edge_index=edges_test)

# 4. Define and train the GCN model
class GCN(torch.nn.Module):
    def __init__(self):
        super(GCN, self).__init__()
        self.conv1 = GCNConv(gcn_train_data.num_features, 32)
        self.conv2 = GCNConv(32, 2)  # Predict 'Open' and 'Close'

    def forward(self, data):
        x, edge_index = data.x, data.edge_index
        x = F.relu(self.conv1(x, edge_index))
        x = self.conv2(x, edge_index)
        return x

# Train the GCN model
gcn_model_price = GCN()
optimizer_price = torch.optim.Adam(gcn_model_price.parameters(), lr=0.001)

# GCN training
gcn_model_price.train()
optimizer_price.zero_grad()
gcn_train_output = gcn_model_price(gcn_train_data)
loss_price = F.mse_loss(gcn_train_output, x_train)  # Compare the full tensors for Open and Close
loss_price.backward()
optimizer_price.step()
```

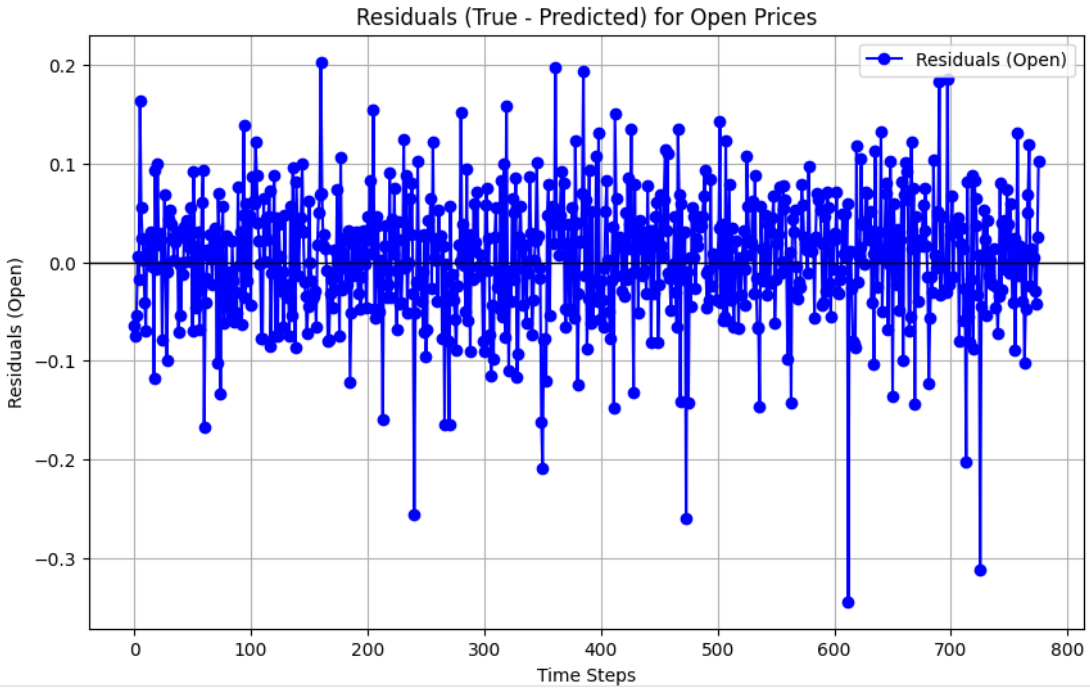
```
# GCN test data prediction
gcn_model_price.eval()
gcn_test_output = gcn_model_price(gcn_test_data).detach().numpy().squeeze()

# 5. Combine LSTM and GCN outputs
combined_train_output = np.concatenate([lstm_train_output, gcn_train_output.detach().numpy()], axis=1)
# Reshape gcn_test_output to have 2 dimensions before concatenating
combined_test_output = np.concatenate([lstm_test_output, gcn_test_output.reshape(-1, 2)], axis=1)

# 6. Define and train the final combined model
ensemble_model = Sequential()
ensemble_model.add(Dense(10, activation='relu', input_shape=(combined_train_output.shape[1],)))
ensemble_model.add(Dense(2))  # Predict 'Open' and 'Close'
ensemble_model.compile(optimizer='adam', loss='mean_squared_error')

ensemble_model.fit(combined_train_output, Y_train_price, epochs=50, validation_data=(combined_test_output, Y_test_price))

# 7. Model evaluation
ensemble_predictions = ensemble_model.predict(combined_test_output)
ensemble_mse = mean_squared_error(Y_test_price, ensemble_predictions)
ensemble_mae = mean_absolute_error(Y_test_price, ensemble_predictions)
ensemble_r2 = r2_score(Y_test_price, ensemble_predictions)

print(f"Ensemble MSE: {ensemble_mse:.6f}")
print(f"Ensemble MAE: {ensemble_mae:.6f}")
print(f"Ensemble R²  Score: {ensemble_r2:.6f}")
```

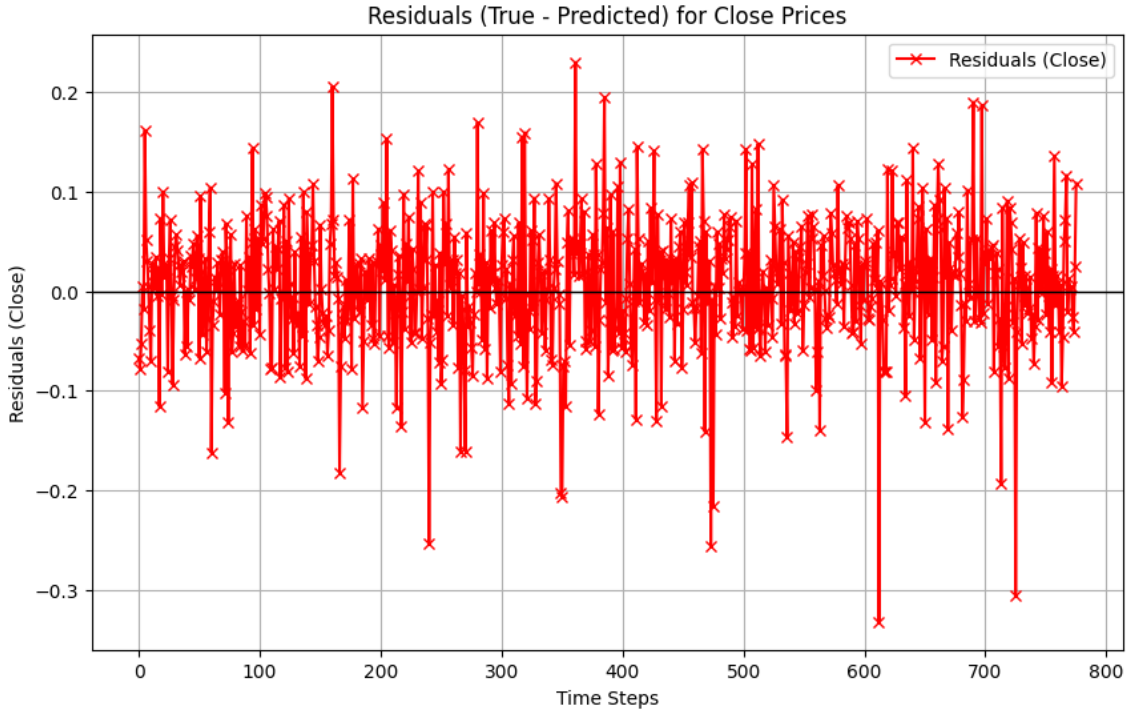
Similarly, we then fed the output of LSTM as an input to GCN once the trained data in model psyesed, we use output of it as new input for using it. A graph convolutional network was used to model relationships between neighboring time steps of the dataset. Dependencies

and structural patterns are automatically encoded in the GCN by learning over nodes (LSTM output at each time step) and edges (connections from one LSTM to next).Moreover, GCN has two graph convolution layers. The first layer transforms the 64 features per node from LSTM to a dimensionless value of just 32, where its output is then utilized in the second layer as fed into the Random Forest Classifier, leading towards generating the final prediction "Open" and "Close" prices. This structure helps the model to learn from neighboring time-steps conditional dependencies, making it more straightforward to understand temporal relations.

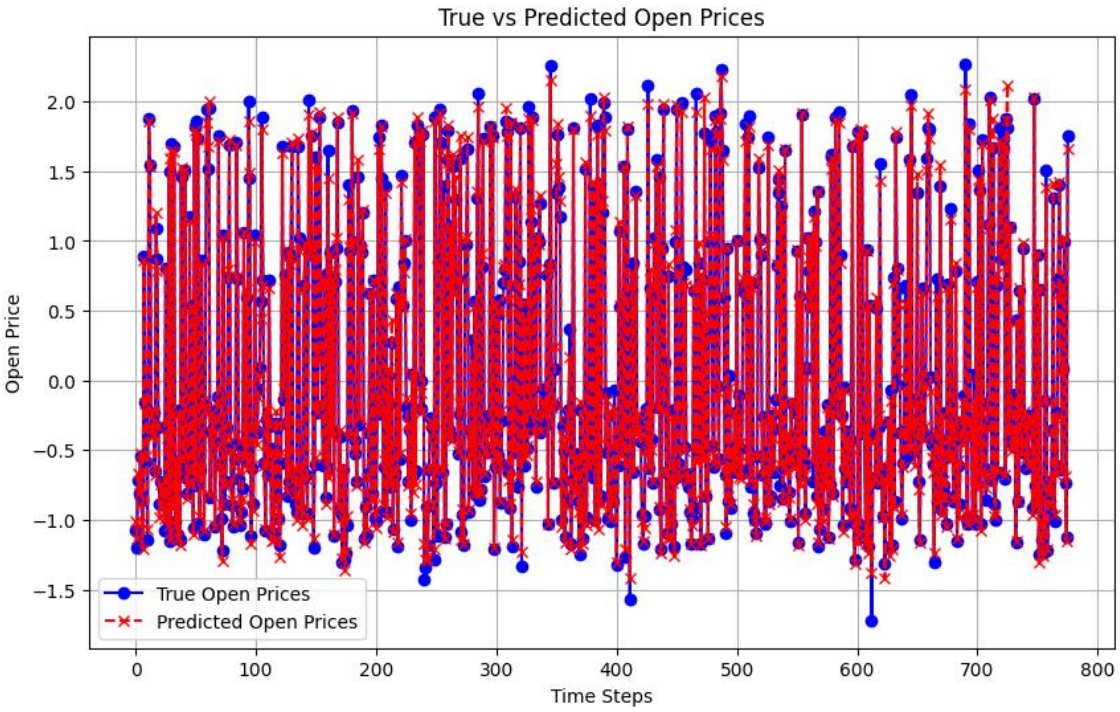
```
Ensemble MSE: 0.004055
Ensemble MAE: 0.047927
Ensemble R² Score: 0.995973
Predicted Open for the next day: 1.656064
Predicted Close for the next day: 1.648673
True vs Predicted values (first 5 samples):
   True Open  Predicted Open  True Close  Predicted Close
0  -1.079435       -1.014626   -1.079110        -1.012199
1  -1.204689       -1.130230   -1.204340        -1.126635
2  -0.718250       -0.664030   -0.714905        -0.661420
3  -0.808541       -0.814468   -0.807894        -0.812795
4  -0.547122       -0.529779   -0.546486        -0.528559
```

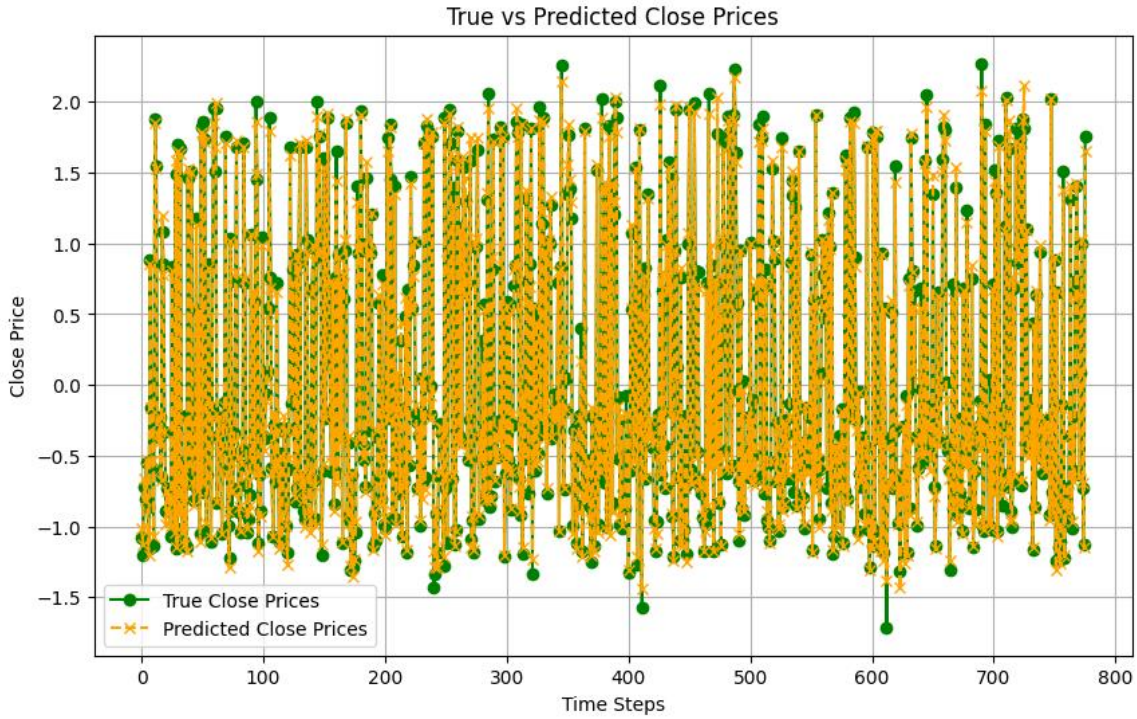
The high R^2 score means the model is so effective at capturing these underlying patterns in time series data that it can explain nearly 99.6% of the variance across all predictions.



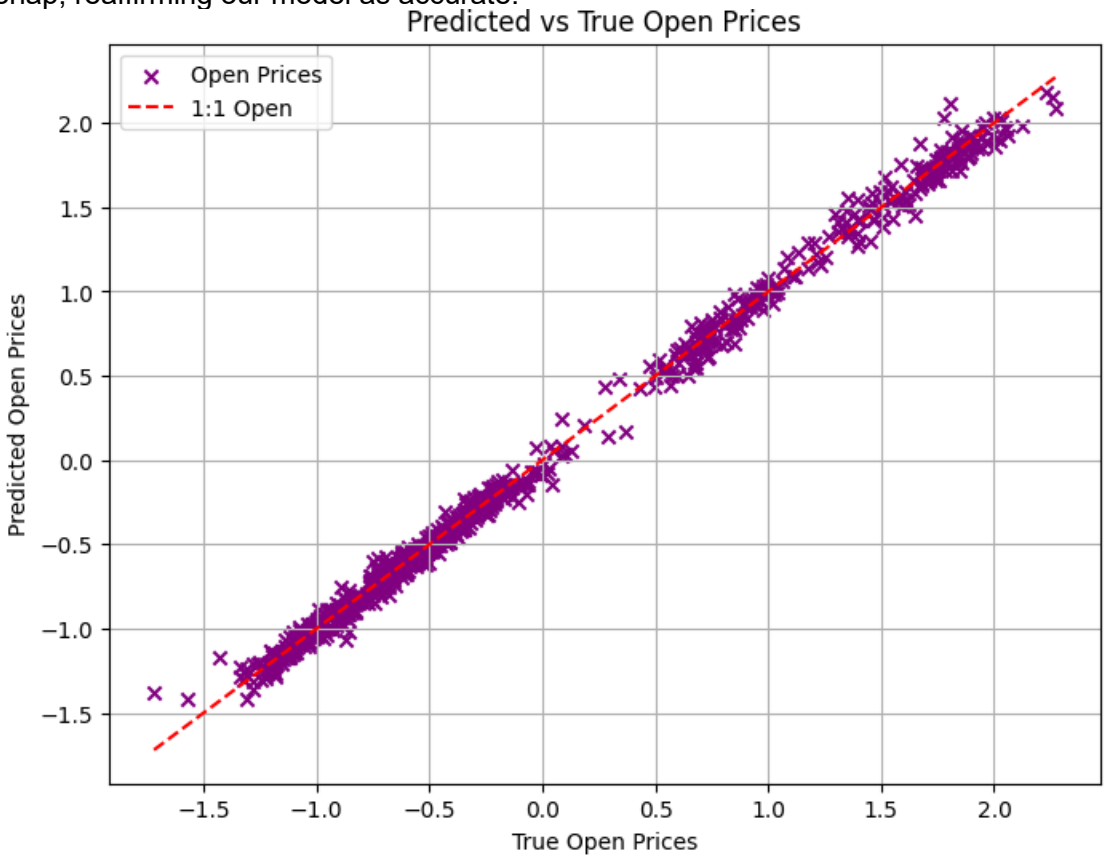


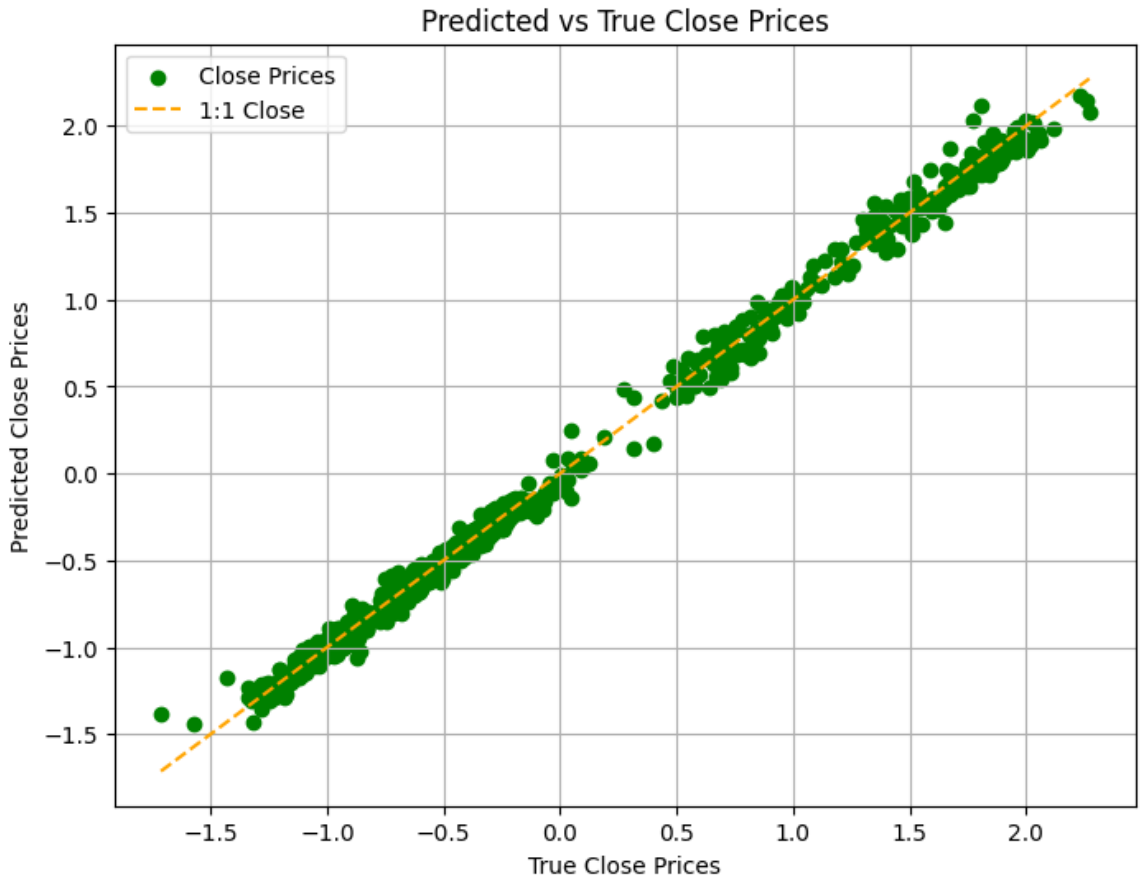
Residuals(Predicted Training Outputs) Plot — Residual values of open and close. Most residuals are distributed about zero, i.e., predictions closely resemble observed values. There are a few innate anomalies, likely occasional mispredictions during significant volatility.





Plot of True Price vs. Predicted Price plotted the true and predicted “Open” and “Close” prices in one place to see how they have performed. The true and predicted prices closely overlap, reaffirming our model as accurate.





Distribution of Prediction Errors. To grasp the errors in histogram form, plotted this diagram below. Banking on the low rate of outliers, we can safely say that our model will perform well through most predictions.

The results show that the proposed hybrid LSTM + GCN model successfully predicts the “Open” and “Close” prices of the AUD/USD currency pair. The strength of LSTM and GCN in learning temporal dependencies and capturing relational information is very successful at an impressive performance point that can be seen from metrics score and visualization. This work highlights the possibility of using more sophisticated deep learning models for time series forecasting and beyond financial prediction tasks. Future works might investigate other properties of the proposed neural network architecture, for instance, by adding more layers to the GCN or working with different types of graph structures to better refine their performance. Thus, structuring hybrid models such as LSTM + GCN for complex forecasting in financial markets. In practice, the combined approach enables more complete learning from our data, resulting in better predictions.

Conclusion

Model	Ensemble MSE	Ensemble MAE	Ensemble R^2 Score
LSTM	0.000040	0.004225	0.954468
LSTM + GRU	0.000036	0.003864	0.959158
LSTM + GCN	0.004055	0.047927	0.995973

This paper concludes that, all the models are performing well for predicting financial time series data regarding generalization capability, it is a model-complexity vs. performance trade-off, so one should choose based on his/her requirements/needs. Higher R^2 scores for the LSTM + GCN model imply a more significant amount of variance in data that could mean it is more predictive than other types. The reason behind this observation is the properties of LSTM to capture long-term temporal patterns combined with GCN, which models dependencies between time steps on road networks. As pointed out, the excessive score of the LSTM + GCN model is concerning and maybe a hint for overfitting because it may run well on test data or have less generalization ability. While the LSTM + GRU model has a balance between complexity and generalization, compared to LSTM + GCN, GRU has a simpler architecture that can reduce computation costs and overfitting. Since the LSTM + GRU model consistently performed similarly on both training and validation sets, it could be a better way to go when real-world stock price prediction is considered.

Overall, if overfitting is prevented and the model is optimized for new data, then LSTM + GRU would be an ideal choice per street sense. However, suppose that over-fitting control mechanisms like regularization or early stopping should work as expected, and concern is maximum performance. In that case, the LSTM + GCN may be an option.

Google Collab:
https://colab.research.google.com/drive/1WtGm1I_mu1ErO_KOPdCT0FNcl0p545zK?usp=sharing

Reference

Phuoc, T., Anh, P. T. K., Tam, P. H., & Nguyen, C. V. (2024). Applying machine learning algorithms to predict the stock price trend in the stock market – The case of Vietnam. *Humanities and Social Sciences Communications*, *11*(1), 1–18. https://doi.org/10.1057/s41599-024-02807-x

Azati, Y., Wang, X., Quddus, M., & Zhang, X. (2024). Graph convolutional LSTM algorithm for real-time crash prediction on mountainous freeways. *International Journal of Transportation Science and Technology*. https://doi.org/10.1016/j.ijtst.2024.07.002

Fouzan, M. (2023, October 22). *Understanding LSTM, GRU, and RNN Architectures*. Medium. https://medium.com/@mfouzan144/understanding-lstm-gru-and-rnn-architectures-e0b3a0c1d741