

# Configuration Manual

MSc Research Project  
Data Analytics

Sahithi Dornala  
Student ID: x23241551

School of Computing  
National College of Ireland

Supervisor: Furqan Rustam

**National College of Ireland**  
**MSc Project Submission Sheet**  
**School of Computing**



**Student Name:** Sahithi Dornala  
**Student ID:** X23241551  
**Programme:** MSc Data Analytics **Year:** 2024 - 25  
**Module:** MSc Research Project  
**Lecturer:** Furqan Rustam  
**Submission Due Date:** 12/12/2024  
**Project Title:** Bitcoin Price Prediction: A Machine Learning Approach Using Opening and Closing Data

**Word Count:** 1067 **Page Count:** 11

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

**Signature:** Sahithi Dornala

**Date:** 12/12/2024

**PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST**

Attach a completed copy of this sheet to each project (including multiple copies)	<input type="checkbox"/> NA
<b>Attach a Moodle submission receipt of the online project submission,</b> to each project (including multiple copies).	<input type="checkbox"/> NA
<b>You must ensure that you retain a HARD COPY of the project,</b> both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	<input type="checkbox"/> NA

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

<b>Office Use Only</b>	
Signature:	
Date:	
Penalty Applied (if applicable):	

# Configuration Manual

Sahithi Dornala

Student ID: x23241551

## 1. Introduction

This manual outlines a step-by-step replication guide for predicting and forecasting the opening and closing prices of Bitcoin using advanced machine learning models. The experiment compares several models, including GRU, LSTM, RNN, Prophet, and Prophet-GRU, whereby GRU surpasses others in accuracy and error minimization. This document lists the packages and configurations of software for the experimental environment so that it can replicate similar results.

## 2. Deployment Environment

The environment used for this research is the local windows operating systems with GPU and CPU on Jupyter Notebook. However, the hardware and the software specification details are mentioned below here:

### 2.1 Hardware Specification

- **Processor:** Intel Core i7 or equivalent
- **RAM:** 16 GB or higher
- **GPU:** NVIDIA RTX 2060 or higher (recommended for faster training of deep learning models).

### 2.2 Software Specification

- **Operating System:** Windows 10/11, macOS, or Linux-based OS
- **Programming Language:** Python 3.11
- **IDE:** Jupyter Notebook or VS Code (with Python extension)

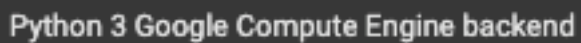


Figure 1: Python Version

#### 2.1.1 Integrated Development Environment (IDE): Jupyter Notebook

### 2.2 Python Libraries Required

Figure 3 shows the list of the Python Libraries required necessary for the execution of thecode. This mentioned python libraries can be installed using the pip command.

- **pandas:** For data manipulation and analysis.
- **NumPy:** For numerical operations, especially array operations.

### Data Visualization Libraries:

- **Seaborn:** For statistical data visualization.
- **Matplotlib:** For creating static, animated, and interactive visualizations.

### Machine Learning Libraries:

- **scikit-learn:** For various machine learning algorithms, including:
  - MinMaxScaler for feature scaling
  - r2\_score and mean\_squared\_error for model evaluation
- **TensorFlow:** For building and training deep learning models, specifically:
  - Sequential model for creating a sequential model
  - Dense layer for fully connected layers
  - LSTM layer for Long Short-Term Memory layers
  - Dropout layer for regularization

```
# Importing essential libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import r2_score, mean_squared_error
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Dropout
```

Figure 2: Libraries Imported

## 3. Data Source

The dataset for this project is sourced from Bitcoin Historical Data. This dataset contains high-frequency Bitcoin price data recorded every minute, including Open, High, Low, Close (OHLC) prices, and trading volumes in Bitcoin and USD.

### Dataset Preparation

#### 1. Accessing the Dataset:

- Download the dataset from the provided Kaggle link. Save it in a .csv format for processing.
- Dataset link: <https://www.kaggle.com/datasets/swaptr/bitcoin-historical-data>

#### 2. Dataset Structure:

- **Timestamp:** Date and time of each record.
- **Open, High, Low, Close Prices:** OHLC prices recorded per minute.
- **Volume (BTC and USD):** Trading volumes for Bitcoin and USD.

#### 3. Data Cleaning and Pre-processing:

- Convert the Timestamp column to datetime format and set it as the index.
- Check for missing or erroneous data and handle it using interpolation or

- removal.
- Normalize the numerical features for stable model training.

#### 4. Project Code Files

- **Data Pre-processing:** Handles data cleaning, normalization, and splitting.
- **Model Implementation:** Includes GRU, LSTM, RNN, Prophet, and Prophet-GRU implementations.
- **Performance Evaluation:** Calculates metrics such as RMSE, MAE, and  $R^2$  for each model.
- **Results Visualization:** Visualizes model predictions against actual prices.

#### 5. Data Preparation

##### 5.1 Data Loading

Load data from CSV file uploaded:

```
# Loading the dataset from a CSV file into a pandas DataFrame
df = pd.read_csv("Kaggle_RIC.csv")

# Slice the DataFrame to include only the first 20,000 rows
df = df[:20000]

# Display the first 5 rows of the DataFrame to get an overview of the data
df.head()
```

	Timestamp	Date	Symbol	Open	High	Low	Close	Volume BTC	Volume USD
0	1676939580000	2023-02-21 00:33:00	BTC/USD	24859.34	24859.34	24859.34	24859.34	0.000000	0.000000
1	1676939520000	2023-02-21 00:32:00	BTC/USD	24821.96	24859.34	24821.96	24859.34	0.103099	2562.977818
2	1676939460000	2023-02-21 00:31:00	BTC/USD	24818.09	24821.96	24815.47	24821.96	0.090640	2249.866178
3	1676939400000	2023-02-21 00:30:00	BTC/USD	24812.25	24818.09	24812.25	24818.09	0.002203	54.681450
4	1676939340000	2023-02-21 00:29:00	BTC/USD	24809.27	24812.25	24809.27	24812.25	0.090675	2249.862431

Figure 3: Dataset Imported

##### 5.2 Data Pre-processing

- **Handling Unnamed Values:** Impute missing data using mean/median or interpolate and dropping unnamed column(s).
- **Date Conversion:** Converting the Date column to required format.

```
# Basic Data Cleaning
# Dropping 'Unnamed: 0' column as it's just an index column

# Convert 'Date' column to datetime format
df['Date'] = pd.to_datetime(df['Date'])

# Set 'Date' as index for time series analysis
df.set_index('Date', inplace=True)
```

Figure 4: Handling duplicate values

## 5.3 EDA and Plotting graphs

```
# Plot of closing price over time
plt.figure(figsize=(10,6))
plt.plot(df.index, df['Close'], label='Close Price', color='blue')
plt.title('Bitcoin Close Price Over Time')
plt.xlabel('Date')
plt.ylabel('Close Price (USD)')
plt.legend()
plt.grid(True)
plt.show()

# Plot volume of Bitcoin traded in BTC and USD over time
fig, ax1 = plt.subplots(figsize=(10,6))

ax1.plot(df.index, df['Volume BTC'], color='orange', label='Volume BTC')
ax1.set_xlabel('Date')
ax1.set_ylabel('Volume BTC', color='orange')

ax2 = ax1.twinx()
ax2.plot(df.index, df['Volume USD'], color='green', label='Volume USD')
ax2.set_ylabel('Volume USD', color='green')

plt.title('Volume of Bitcoin Traded in BTC and USD Over Time')
fig.tight_layout()
plt.grid(True)
plt.show()
```

Figure 5: Visualizing Bitcoin price over time

## 6. Model Building

- GRU (Gated Recurrent Units)

```
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, GRU
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
from math import sqrt

# Extracting 'Close' price for time series forecasting
close_prices = df['Close'].values.reshape(-1, 1)

# Normalizing the Close prices using MinMaxScaler
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_close_prices = scaler.fit_transform(close_prices)

# Splitting data into training and test sets (80% train, 20% test)
train_size = int(len(scaled_close_prices) * 0.8)
train_data = scaled_close_prices[:train_size]
test_data = scaled_close_prices[train_size:]

# Create datasets for GRU (using past 60 time steps to predict the next time step)
def create_dataset(data, time_step=60):
    X, y = [], []
    for i in range(len(data)-time_step):
        X.append(data[i:i+time_step, 0])
        y.append(data[i+time_step, 0])
    return np.array(X), np.array(y)

time_step = 60
X_train, y_train = create_dataset(train_data, time_step)
X_test, y_test = create_dataset(test_data, time_step)

# Reshaping the data for GRU [samples, time steps, features]
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)

# Building the GRU model
gru_model = Sequential([
    GRU(50, return_sequences=True, input_shape=(X_train.shape[1], 1)),
    GRU(50),
    Dense(1)
])
gru_model.compile(optimizer='adam', loss='mean_squared_error')

# Fit the model
gru_model.fit(X_train, y_train, epochs=10, batch_size=72, validation_data=(X_test, y_test), verbose=2)

# Make predictions
gru_predictions = gru_model.predict(X_test)

# Calculate metrics
gru_rmse = sqrt(mean_squared_error(y_test, gru_predictions))
gru_r2 = r2_score(y_test, gru_predictions)
gru_mae = mean_absolute_error(y_test, gru_predictions)
```

Figure 6: Model training code snippet for GRU

- **LSTM**

```
# Extracting 'Open' price for time series forecasting
open_prices = df['Open'].values.reshape(-1, 1)

# Normalizing the close prices using MinMaxScaler
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_open_prices = scaler.fit_transform(open_prices)

# Splitting data into training and test sets (80% train, 20% test)
train_size = int(len(scaled_open_prices) * 0.8)
train_data = scaled_open_prices[:train_size]
test_data = scaled_open_prices[train_size:]

time_step = 60
X_train, y_train = create_dataset(train_data, time_step)
X_test, y_test = create_dataset(test_data, time_step)

# Reshaping the data for LSTM [samples, time steps, features]
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)

# Build the LSTM model
model = Sequential()
model.add(LSTM(units=50, return_sequences=True, input_shape=(X_train.shape[1], 1)))
model.add(Dropout(0.2))
model.add(LSTM(units=50, return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(units=1))

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')

# Display a summary of the Keras model
model.summary()

# Train the model
history = model.fit(X_train, y_train, epochs=10, batch_size=64, verbose=1)

# Make predictions
train_predict = model.predict(X_train)
test_predict = model.predict(X_test)

# Inverse transform predictions and actual values to original scale
train_predict = scaler.inverse_transform(train_predict)
test_predict = scaler.inverse_transform(test_predict)
y_train_inv = scaler.inverse_transform(y_train.reshape(-1, 1))
y_test_inv = scaler.inverse_transform(y_test.reshape(-1, 1))

# Calculate R^2 and RMSE for the test set
r2 = r2_score(y_test_inv, test_predict)
rmse = np.sqrt(mean_squared_error(y_test_inv, test_predict))

print(f"R^2 Score: {r2}")
print(f"RMSE: {rmse}")
```

Figure 7: LSTM Model Training

- **Prophet**

Use the Prophet library for time-series forecasting:

```
import pandas as pd
from prophet import Prophet
import math
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
from sklearn.preprocessing import MinMaxScaler

# Extracting 'Open' price for time series forecasting
open_prices = df['Open'].values.reshape(-1, 1)

# Normalizing the close prices using MinMaxScaler
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_open_prices = scaler.fit_transform(open_prices)

# Splitting data into training and test sets (80% train, 20% test)
train_size = int(len(scaled_open_prices) * 0.8)
train_data = scaled_open_prices[:train_size]
test_data = scaled_open_prices[train_size:] # Assign test_data

# Prepare data for Prophet - convert index to datetime
df_prophet = pd.DataFrame(scaled_open_prices, columns=['y']) # Use scaled_open_prices
df_prophet['ds'] = df.index # Use the full df index

# Splitting data based on the same index used before
train_df = df_prophet[:train_size]
test_df = df_prophet[train_size:]

# Initialize and fit the model
prophet_model = Prophet(daily_seasonality=False)
prophet_model.fit(train_df)

# Make predictions
future = prophet_model.make_future_dataframe(periods=len(test_df))
forecast = prophet_model.predict(future)

# Get predictions for the test set
prophet_predictions = forecast['yhat'][-len(test_df):]

# Evaluate
prophet_rmse = math.sqrt(mean_squared_error(test_df['y'], prophet_predictions))
prophet_r2 = r2_score(test_df['y'], prophet_predictions)
prophet_mae = mean_absolute_error(test_df['y'], prophet_predictions)

print(prophet_mae, prophet_r2, prophet_rmse)
```

Figure 8: Prophet Model Training

- **Prophet-GRU**

Combine GRU predictions with Prophet forecasts for hybrid modelling.

```
import numpy as np
import pandas as pd
from prophet import Prophet
from keras.models import Sequential
from keras.layers import Dense, GRU
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Assuming df is already loaded with a proper datetime index
if not df.index.is_monotonic_increasing:
    df = df.sort_index() # Ensure the index is sorted

# Preparing the 'Open' price data
open_prices = df['Open'].values.reshape(-1, 1)
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_open_prices = scaler.fit_transform(open_prices)

# Define dataset creation function
def create_dataset(data, time_step):
    X, y = [], []
    for i in range(time_step, len(data)):
        X.append(data[i-time_step:i, 0])
        y.append(data[i, 0])
    return np.array(X), np.array(y)

# Train-test split
train_size = int(len(scaled_open_prices) * 0.8)
train_data = scaled_open_prices[:train_size]
test_data = scaled_open_prices[train_size:]

time_step = 60
X_train, y_train = create_dataset(train_data, time_step)
X_test, y_test = create_dataset(test_data, time_step)

# Reshaping for GRU
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)

# GRU model
model = Sequential([
    GRU(50, return_sequences=True, input_shape=(time_step, 1)),
    GRU(50),
    Dense(1)
])
model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(X_train, y_train, epochs=10, batch_size=32)

# Prediction
predicted_stock_price = model.predict(X_test)
predicted_stock_price = scaler.inverse_transform(predicted_stock_price)

# Prepare Prophet DataFrame
forecast_dates = pd.date_range(start=df.index[train_size + time_step], periods=len(predicted_stock_price), freq='B')
df_prophet = pd.DataFrame({
    'ds': forecast_dates,
    'y': predicted_stock_price.flatten()
})

# Prophet model
prophet_model = Prophet(daily_seasonality=True)
prophet_model.fit(df_prophet)
future = prophet_model.make_future_dataframe(periods=30)
forecast = prophet_model.predict(future)

# Plotting
fig = prophet_model.plot(forecast)

# Calculating metrics
true_prices = scaler.inverse_transform(test_data[time_step:].reshape(-1, 1))
rmse = np.sqrt(mean_squared_error(true_prices, predicted_stock_price))
mae = mean_absolute_error(true_prices, predicted_stock_price)
r2 = r2_score(true_prices, predicted_stock_price)

print(f"R-squared: {r2:.3f}")
print(f"RMSE: {rmse:.3f}")
print(f"MAE: {mae:.3f}")
```

Figure 9: Prophet and GRU Model Training

- **RNN Model**



```

import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, SimpleRNN
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
from math import sqrt

# Extracting 'Close' price for time series forecasting
close_prices = df['Close'].values.reshape(-1, 1)

# Normalizing the close prices using MinMaxScaler
scaler = MinMaxScaler(feature_range=(0, 1))
scaled_close_prices = scaler.fit_transform(close_prices)

# Splitting data into training and test sets (80% train, 20% test)
train_size = int(len(scaled_close_prices) * 0.8)
train_data = scaled_close_prices[:train_size]
test_data = scaled_close_prices[train_size:]

# Create datasets for RNN (using past 60 time steps to predict the next time step)
def create_dataset(data, time_step=60):
    X, y = [], []
    for i in range(len(data)-time_step):
        X.append(data[i:i+time_step, 0])
        y.append(data[i+time_step, 0])
    return np.array(X), np.array(y)

time_step = 60
X_train, y_train = create_dataset(train_data, time_step)
X_test, y_test = create_dataset(test_data, time_step)

# Reshaping the data for RNN [samples, time steps, features]
X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
X_test = X_test.reshape(X_test.shape[0], X_test.shape[1], 1)

# Build the RNN model
rnn_model = Sequential([
    SimpleRNN(50, return_sequences=True, input_shape=(X_train.shape[1], 1)),
    SimpleRNN(50),
    Dense(1)
])
rnn_model.compile(optimizer='adam', loss='mean_squared_error')

# Fit the model
rnn_model.fit(X_train, y_train, epochs=10, batch_size=72, validation_data=(X_test, y_test), verbose=2)

# Make predictions
rnn_predictions = rnn_model.predict(X_test)

# Calculate metrics
rnn_rmse = sqrt(mean_squared_error(y_test, rnn_predictions))
rnn_r2 = r2_score(y_test, rnn_predictions)
rnn_mae = mean_absolute_error(y_test, rnn_predictions)

print(rnn_mae, rnn_r2, rnn_rmse)

```

Figure 10: Training Code for RNN

## 7.3 Evaluation

Metrics Calculated:

- Mean Squared Error (MSE)
- Mean Absolute Error (MAE)
- R<sup>2</sup> Score

```

gru_rmse, gru_r2, gru_mae

(0.004882852406880235, 0.9882579073809152, 0.003402596114998774)

# Calculate R² and RMSE for the test set
r2 = r2_score(y_test_inv, test_predict)
rmse = np.sqrt(mean_squared_error(y_test_inv, test_predict))

print(f"R² Score: {r2}")
print(f"RMSE: {rmse}")

R² Score: 0.9850602120388922
RMSE: 29.99198962857897

print(prophet_mae, prophet_r2, prophet_rmse)

01:12:30 - cmdstanpy - INFO - Chain [1] start processing
01:12:40 - cmdstanpy - INFO - Chain [1] done processing

24.808223960786144 -160545.68176593553 28.487903564165098

print(f"R-squared: {r2:.3f}") #Prophet-GRU
print(f"RMSE: {rmse:.3f}")
print(f"MAE: {mae:.3f}")

R-squared: 0.982
RMSE: 22.702
MAE: 18.793

print(rnn_mae, rnn_r2, rnn_rmse)

0.005946663075377108 0.9886076662998037 0.0069044759237693425

```

Figure 9: Results for models

## 7. Results and Visualizations

**Best Model:** GRU consistently outperformed all other models, achieving near-perfect  $R^2$  scores for both opening and closing prices.

**Prophet Model:** Prophet struggled with the high-frequency financial data, demonstrating significantly lower accuracy and higher error rates due to the lack of significant seasonality in Bitcoin price trends.

**Prophet-GRU Hybrid:** Despite the hypothesis that combining Prophet's trend detection and GRU's sequential learning could enhance performance, Prophet-GRU could not outperform standalone GRU.

Model	Target	$R^2$ Score	RMSE	MAE
GRU	Opening Price	0.9961	0.0039	0.0024
GRU	Closing Price	0.9882	0.0048	0.0034
Prophet	Opening Price	0.-160545.682	24.8082	28.4879
Prophet	Closing Price	-154211.521	24.3558	27.9656
Prophet-GRU	Opening Price	0.990	17.023	11.650
Prophet-GRU	Closing Price	0.926	46.699	44.172

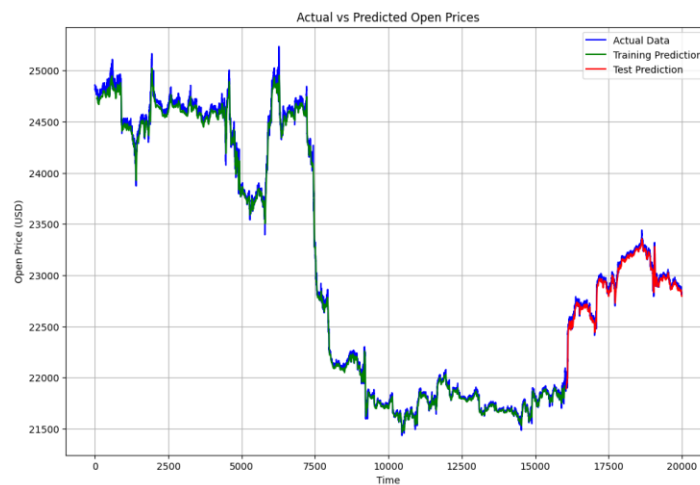


Figure 10: Plots for Forecast

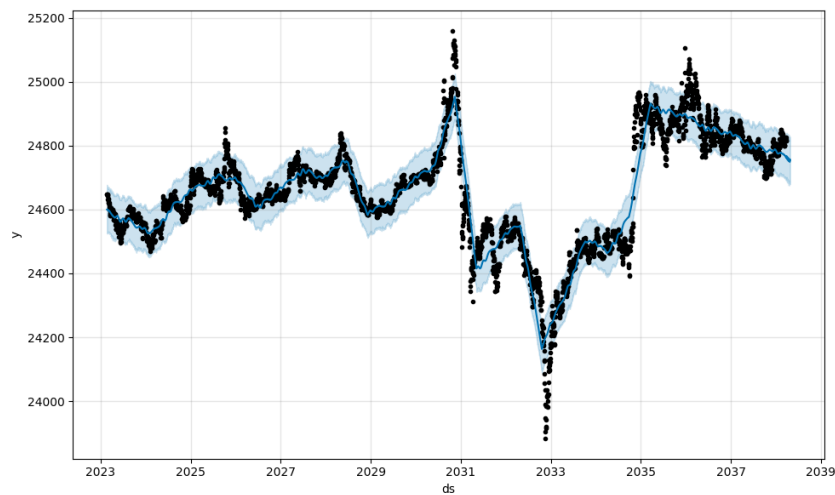


Figure 10: Combined model results GRU and PROPHET

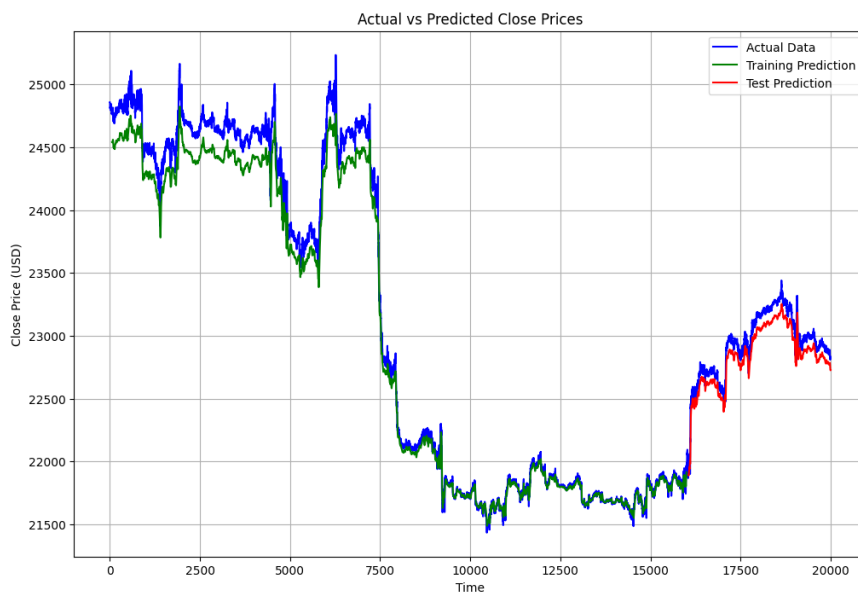


Figure 11 : Forecast Plot

### Key Takeaways

- GRU is the most effective model for forecasting Bitcoin prices, excelling in high-frequency data.
- Prophet's limitations in handling non-seasonal data emphasize the importance of choosing models suited for the dataset characteristics.
- Future work could explore advanced ensemble methods or hybrid approaches to improve predictions further.