Bitcoin Prediciton Project

1st Dimitrios Tsalapatas - 03246
Electircal and Computer Enginnering
University of Thessaly
Volos, Greece
dtsalapatas@e-ce.uth.gr

2nd Ioannis Reinos - 03390
Electircal and Computer Enginnering
University of Thessaly
Volos, Greece
ireinos@e-ce.uth.gr

3rd Eleni Athanailidi - 03453
Electircal and Computer Enginnering
University of Thessaly
Volos, Greece
eathanailidi@e-ce.uth.gr

4th Georgios Kapakos - 03165
Electircal and Computer Enginnering
University of Thessaly
Volos, Greece
gkapakos@e-ce.uth.gr

I. Introduction

Cryptocurrencies have gained significant attention over the past decade, offering a decentralized and digital alternative to traditional financial systems. Among them, Bitcoin stands out as the most widely recognized and valuable cryptocurrency [1]. However, its highly volatile nature makes price prediction a challenging task, attracting interest from researchers and investors alike.

In this project, we develop a neural network-based model to predict the daily price of Bitcoin. Various approaches have been proposed for time series forecasting, ranging from statistical models like LSTM to advanced deep learning techniques. Our goal is to leverage a neural network architecture to forecast Bitcoin prices.

II. EXPLORATORY DATA ANALYSIS (EDA)

The dataset used in this project consists of Bitcoin price data (in USD), spanning from January 1, 2021, at 12:01 AM to March 1, 2022, at 3:43 AM. The data is recorded at a minute-level granularity and includes key trading metrics for each timestamp.

Each row in the dataset contains the following columns:

- **Date**: The timestamp of the recorded data in `MM-DD-YY HH:MM` format.
- **Symbol**: The trading pair, which in this case is BTC/USD.
- **Open**: The Bitcoin price at the beginning of the recorded minute.
- **High**: The highest price of Bitcoin during that minute.
- **Low**: The lowest price of Bitcoin during that minute.
- **Close**: The closing price of Bitcoin at the end of the recorded minute.
- **Volume BTC**: The amount of Bitcoin traded during that minute.
- **Volume USD**: The equivalent trading volume in US dollars.

This dataset provides a detailed view of Bitcoin's price fluctuations and trading activity, making it suitable for time series forecasting. The minute-level granularity allows for a high-resolution analysis of market trends and volatility. For our experiments, we preprocess and aggregate the data as necessary to train our models effectively.

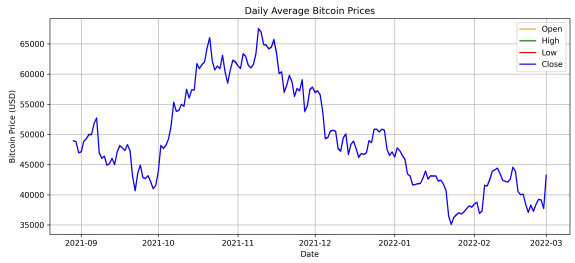

Fig. 1. Daily Average open, high, low, close prices of bitcoin

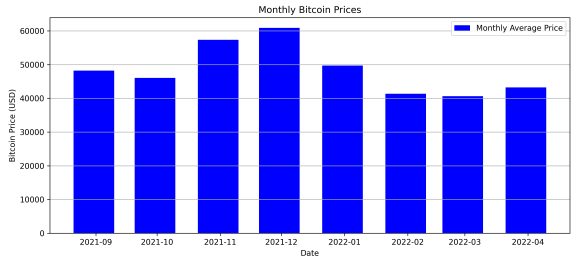

Fig. 2. Monthly value of bitcoin

Fig 2. presents the daily trends. Bitcoin peaked above $65,000 in late 2021 before declining into early 2022. The lowest price, under $35,000, occurred in March 2022 before a minor recovery

Fig 3. shows the average monthly Bitcoin prices. The price peaked in December 2021 at over $60,000, followed by a decline into early 2022, stabilizing near $40,000 by April.

III. Data Preprocessing

Before training the neural network model, several preprocessing steps were applied to clean and prepare the dataset [2]. We tried three different implementation techniques for preprocessing, one that target is the close value of each minute (as it is on dataset), one that uses the average close value of each day, and finally one that the target is the transition of bitcoin price. For all cases we did the following: The *date* column was converted into a `datetime` format to facilitate time-based operations. Additionally, the dataset was sorted in ascending order based on the *date* column to ensure a proper chronological sequence. After this operation, the dataset index was reset to ensure a continuous sequence. Certain columns were removed as they did not contribute significant information to the prediction task:

- **unix**: A Unix timestamp, redundant since the *date* column was already available.
- **symbol**: The trading pair, which was always BTC/USD, making it unnecessary.
- **Volume BTC** and **Volume USD**: These columns were removed to focus on price movements rather than trading volume.

A. 1^{st} *Implementation: Close value per minute*

In this implementation, our target variable is the actual closing price of Bitcoin for each minute. We aim to predict the *close* value at the end of each minute, as it represents the final recorded price within that time frame. To enhance model performance and ensure stable training, we normalize the target variable using MinMax scaling, transforming the *close* values to a range between 0 and 1. This normalization helps improve convergence and overall prediction accuracy.

Here is the plot of bitcoin values after the MinMax scaling:

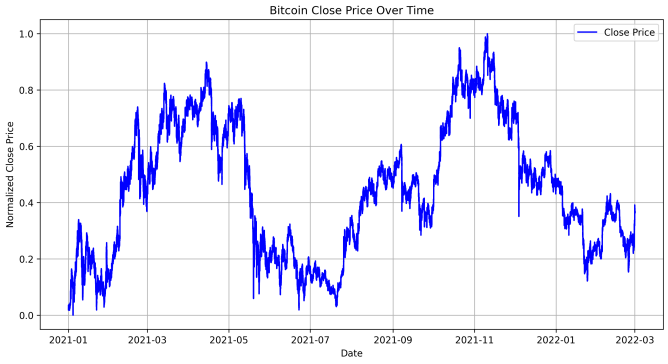

Fig. 3. Close value per minute after MinMax scale

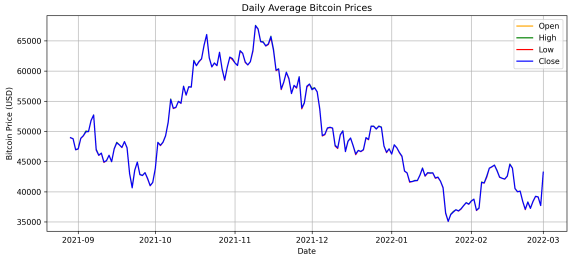

Fig. 4. Daily Average open, high, low, close prices of bitcoin

Fig. 5. Monthly value of bitcoin

B. 2^{nd} Implementation

In this approach, instead of predicting the close value per minute, we aimed to predict the close value per day. To achieve this, we computed the daily average of all minute-level close values for each day. This transformation smooths out short-term fluctuations and provides a more stable trend for forecasting.

Advantages:

- Reduces noise by averaging out minute-level volatility.
- Decreases data volume, improving computational efficiency.

Disadvantages:

- Loses fine-grained details of minute-level price movements.
- May not capture sudden intraday market changes effectively.

Here we present the plot of daily average close value:

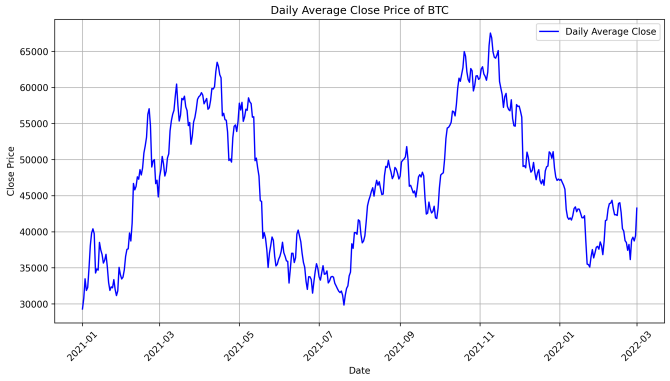

Fig. 6. Daily average close values

C. 3^{rd} Implementation

After researching cryptocurrency and its price prediction, we realized that accurately predicting the exact value of Bitcoin (BTC) is not correct approach. Instead of focusing on predicting the absolute price, a more effective approach is to identify trends in the dataset. Therefore, in this implementation, our target variable is the price change rather than the actual price. Specifically, we predict the difference between the closing price of the next minute and the closing price of the current minute. This approach helps capture market trends and short-term fluctuations more effectively.

This data preprocessing step extracts various date-related features from the dataset, helping the model capture temporal patterns that may influence price movements. By including these features, the model can learn seasonal trends and periodic behaviors, improving its predictive performance. The extracted features are:

- **Day**: The day of the month (1–31).
- **Day of the week**: Represents the weekday (0 for Monday, 6 for Sunday).
- **Day of the year**: The ordinal day in the year (1–365 or 1–366 in leap years).
- **Is month end**: A binary indicator (1 if the date is the last day of the month, else 0).
- **Is month start**: A binary indicator (1 if the date is the first day of the month, else 0).
- **Is quarter end**: A binary indicator (1 if the date is the last day of a quarter, else 0).
- **Is quarter start**: A binary indicator (1 if the date is the first day of a quarter, else 0).
- **Is year end**: A binary indicator (1 if the date is the last day of the year, else 0).
- **Is year start**: A binary indicator (1 if the date is the first day of the year, else 0).

By incorporating these features, the model can better understand seasonal trends and make more informed predictions.

To enhance model performance, the dataset was optionally enriched with technical indicators in order to provide the trend of price move of BTC on our model.

- **Relative Strength Index (RSI)**: A momentum indicator that measures overbought or oversold conditions.
- **Exponential Moving Averages (EMA)**: Short-term (*EMAF* with length 20) and medium-term (*EMAM* with length 50) trends.
- **Moving Average Convergence Divergence (MACD)**: Used to assess trend strength.

D. Additional Preprocessing Methods

To enhance the robustness of the data, several additional preprocessing techniques were applied.

1) Lag Features: Lag features are created by shifting the target variable (in this case, `Close`) by a specified number of time steps. This allows the model to learn from past observations when making future predictions. By incorporating lagged values, the model can capture patterns and temporal dependencies that are crucial for time-series forecasting.

For example, a lag feature such as `close_lag_1` represents the close value 1 time step ago. These lag features enable the model to understand how past close values influence future close values, improving predictive accuracy [3].

2) Rolling Window: The rolling window technique [4] involves computing moving averages of the target variable over a specified time window. This approach helps smooth out short-term fluctuations in the data, making it easier for the model to capture long-term trends. For instance, a feature such as `close_ma_5` represents the average close value over the last 5 time steps. By leveraging rolling windows, the model learns from smoothed, long-term patterns rather than reacting to minor fluctuations. This technique is particularly beneficial for stabilizing predictions and improving robustness.

3) Significance of Lag Features and Rolling Windows: Both lag features and rolling windows contribute significantly to the model's predictive performance. Lag features allow the model to identify historical trends and understand how previous close value affect future values. Meanwhile, rolling windows help reduce noise and highlight meaningful patterns in the data. By integrating these techniques, the model gains

a more comprehensive understanding of the data, leading to improved prediction accuracy and stability.

4) Handling Missing Values: Handling missing values is a critical step in data preprocessing [5], as models often perform poorly when the dataset contains `NaN` values. Two potential approaches were considered:

1) Removing rows with missing values.
2) Replacing missing values with the column mean.

The second approach was chosen, as dropping rows results in data loss, which is particularly problematic in time-series forecasting where maintaining the temporal structure is essential.

5) Handling Outliers: Managing outliers is essential to ensure accurate model predictions. We used the IQR method.

6) Interquartile Range (IQR) Method: The IQR method identifies outliers by examining the range between the first quartile (Q1) and the third quartile (Q3). The IQR is defined as:

$$IQR = Q3 - Q1$$

Outliers are defined as data points that fall outside the range:

$$[Q1 - 1.5 \times IQR,\ Q3 + 1.5 \times IQR]$$

This method is robust to skewed data and extreme values, making it widely applicable. It removes points that are unusually far from the central distribution, improving the accuracy of models by focusing on representative data.

IV. ARIMA Implementation

A. What is ARIMA?

ARIMA (AutoRegressive Integrated Moving Average) [6] is a widely used model for time series forecasting. It's powerful because it captures patterns in past data to predict the future. The name itself hints at its three main components:

- **Autoregression (AR)**: Relies on past values to predict future ones.
- **Differencing (I)**: Removes trends to make the data stationary.
- **Moving Average (MA)**: Captures past forecasting errors to improve predictions.

An ARIMA model is denoted as **ARIMA(p, d, q)**, where:

- **p** = Number of past observations (lags) used.
- **d** = Number of differencing steps to achieve stationarity.
- **q** = Size of the moving average window.

B. Making the Data Stationary

A stationary time series is crucial for ARIMA. We check this using the **Augmented Dickey-Fuller (ADF) test**. If the p-value is greater than 0.05, we need to apply differencing:

- First-order differencing: Subtract each value from the previous one.
- If still non-stationary, apply second-order differencing.
- In our case, first-order differencing was enough.

C. Choosing ARIMA Parameters

To find the best values for **p** and **q**, we analyze:

- **Autocorrelation Function (ACF)**: Helps determine q.
- **Partial Autocorrelation Function (PACF)**: Helps determine p.
- Based on our plots:
  - PACF suggests **p = 1**.
  - ACF suggests **q = 1**.
  - Since differencing was necessary, **d = 1**.

D. Training and Testing the Model

We split the data:

- **Training set**: Data before February 19, 2022.
- **Testing set**: The last 10 days of February 2022.
- We use the 'close' price as our target variable.

Using the chosen parameters, we train an **ARIMA(0,1,1)** model. Once trained, we forecast the next 10 days and reverse the differencing step to return the predictions to the original scale.

E. Evaluating the Model

To measure accuracy, we compare predicted vs. actual prices:

- Plot predictions alongside real values.
- Analyze residuals to check for patterns
- Calculate error metrics:
  - **RMSE (Root Mean Squared Error)**: Measures overall prediction error.
  - **MAE (Mean Absolute Error)**: Captures average deviation.
  - **MAPE (Mean Absolute Percentage Error)**: Represents error in percentage terms.
  - **R^2 Score**: Measures how well our model explains variance in data.

F. Results

TABLE I
RESULTS OF ARIMA

Evaluation of ARIMA	
R2	-2.27
MSE	1889.85
MAE	1584.23

Fig. 7. justification

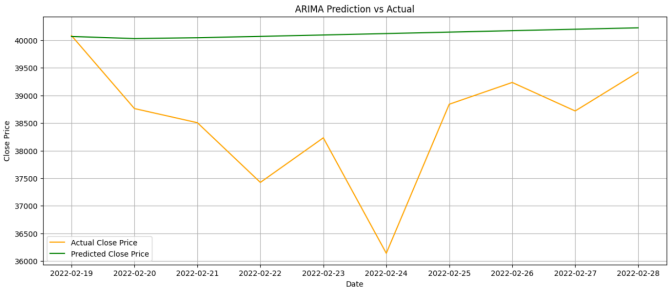

Fig. 8. ARIMA prediction vs. real values for the last 10 days. `ARIMA_Predictions_vs_Real`.

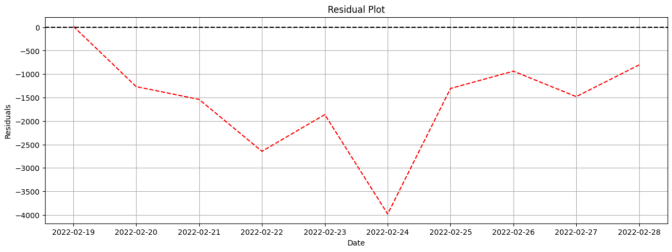

Fig. 9. justification

Fig. 10. ARIMA Residuals of the last 10 days. `ARIMA_Residuals`.

V. LSTM MODEL IMPLEMENTATION

In this section, we describe our implementation of the Long Short-Term Memory (LSTM) neural network for Bitcoin price prediction [7]. We implemented this neural network using Tensorflow [8] python. LSTMs are a type of recurrent neural network (RNN) specifically designed to handle sequential data and long-term dependencies, making them well-suited for time series forecasting.

A. LSTM Network Architecture

This model is a simple yet effective deep learning architecture designed for time-series forecasting using Long Short-Term Memory (LSTM) networks [9]. It consists of a sequential model with an LSTM layer containing 50 units and a ReLU activation function, which helps capture temporal dependencies in sequential data. The LSTM layer processes input sequences of shape $(\text{time steps}, 1)$, allowing the model to learn patterns from past values. A Dense layer with a single neuron follows, serving as the output layer to predict the next closing value of Bitcoin. The model is compiled using the Adam optimizer, which provides efficient gradient-based optimization, and mean squared error (MSE) as the loss function, ensuring that it minimizes the difference between predicted and actual values. This architecture is well-suited for time-series forecasting, leveraging LSTM's ability to remember long-term dependencies while maintaining computational efficiency.

B. Training and Evaluation

For the evaluation of our model, we conducted experiments with three different preprocessing implementations. The first step involved splitting the dataset into two subsets: the `train data` and the `test data`. The `test data` consists of observations from 18/02 to 28/02, while the `train data` includes all data points prior to the test period. This partitioning allows for an unbiased evaluation of the model's performance on unseen data.

We experimented with different numbers of epochs and batch sizes for each implementation to determine the optimal configuration. The number of epochs influences how many times the model will iterate over the entire training dataset, while the batch size determines the number of samples the model processes before updating the weights. In general, using a larger number of epochs typically leads to better performance, as the model has more opportunities to learn from the data. However, this comes at the cost of increased training time.

The batch size plays an important role in the model's training process. A smaller batch size can result in noisier updates to the model's weights, but it may lead to faster convergence. Larger batch sizes, on the other hand, provide more stable updates but may require more memory and take longer to process each iteration. Therefore, a balance must be struck between batch size and training runtime.
Here we can see the runtime for different batch sizes and for 10 epochs, and we measure the runtime and the accuracy.

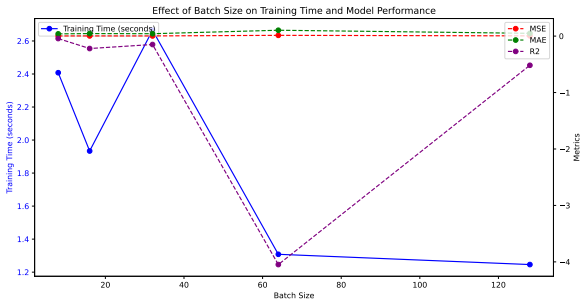

Fig. 11. Runtime for different batch sizes

After training, we evaluated our model's performance using **Root Mean Squared Error (RMSE)** and visually compared predicted vs. actual prices to assess its accuracy. These evaluation metrics helped us fine-tune the model for better forecasting results.

C. Evaluation Metrics

In most studies, four metrics are used to evaluate the precision of forecasting: RMSE, MAPE, R^2, and MAE [10], [11]. Each metric highlights different aspects of the error. However, in the most cases the most important metric for describing the predictive power of a model is considered R^2. Below are the descriptions of the four most commonly used metrics:

- **Root Mean Squared Error (RMSE)**: This metric measures the average magnitude of the errors between predicted and actual values. It is calculated as the square root of the average squared differences between prediction and actual values:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

  A lower RMSE value indicates a better fit to the data.
- **Mean Absolute Percentage Error (MAPE)**: This metric expresses the error as a percentage, making it easier to interpret. It is calculated as the average of the absolute percentage differences between actual and predicted values:

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_i - \hat{y}_i}{y_i}\right| \times 100$$

  While MAPE is widely used, it can be sensitive to very small values of y_i, leading to inflated errors.
- **R-Squared** (R^2): This metric quantifies the proportion of variance in the target variable that is explained by the model. An R^2 value closer to 1 indicates that the model explains most of the variance, while a value closer to 0 means the model is not explaining much of the variance.

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

  R^2 is widely regarded as one of the most important metrics for evaluating the predictive power of a model.
- **Mean Absolute Error (MAE)**: This metric calculates the average magnitude of the errors in the predictions, without considering their direction (positive or negative). It is calculated as:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

  MAE is straightforward to interpret and gives a simple measure of the average error.

D. Results for each implementation

The models were trained on Google Colab, utilizing a cloud-based service powered by an Intel Xeon CPU with 2 virtual CPUs (vCPUs) and 13GB of RAM. So the following runtime is based on this hardware.

TABLE II
RESULTS OF IMPLEMENTATION 1

Evaluation of Implementation 1	
R2	**0.99**
MSE	**2.31e-6**
MAE	**0.0012**
Epochs	**10**
Batch Size	**64**
Training Time (sec)	**677.90**
Inference Time (sec)	**0.000108**

The first implementation achieved excellent performance in terms of predictive accuracy, with an R^2 score of **0.99**, indicating that the model explains nearly all the variance in the data. The **Mean Squared Error (MSE)** is very low at **2.31e-6**, and the **Mean Absolute Error (MAE)** of **0.0012** further confirms the model's precision in predicting Bitcoin's closing price.

The model was trained for **10 epochs** with a **batch size of 64**, resulting in a total training time of approximately **677.90 seconds**. The inference time per prediction was extremely fast, at just **0.000108 seconds**, making it highly efficient for real-time applications.

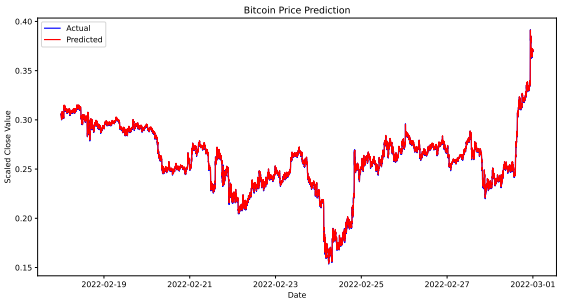

Fig. 12. Plot for actual and predicted values - Implementation 1

However, despite the high accuracy, this implementation is not entirely appropriate for Bitcoin price prediction from an economic perspective. The model focuses on predicting the actual price values, which are large numbers, instead of capturing the price movement or trend. In financial modeling, predicting price changes rather than absolute values is generally more meaningful, as it aligns better with trading strategies and economic analysis. Therefore, while this model performs

well in terms of numerical accuracy, it does not fully meet the requirements for effective Bitcoin price forecasting.

TABLE III
RESULTS OF IMPLEMENTATION 2

Evaluation of Implementation 2	
R2	0.21
MSE	0.002
MAE	0.034
Epochs	100
Batch Size	32
Training Time (sec)	18.90
Inference Time (sec)	0.00315

This implementation, which predicts the **average daily closing price**, aligns better with economic principles as it focuses on general price movement rather than absolute values. However, despite being a more appropriate approach, it is still not entirely correct for financial modeling.

The model's R^2 **score of 0.21** indicates weak predictive power, meaning it does not capture much of the variance in the data. The **MSE (0.002)** and **MAE (0.034)** further suggest that the model has a relatively high error. While the training process was efficient, completing **100 epochs** in just **18.90 seconds**, the overall performance remains suboptimal.

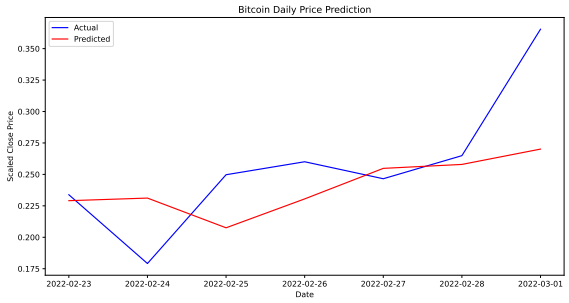

Fig. 13. Plot of actual and predicted values - Implementation 2

Although this method is more suitable for economic analysis, it still requires refinement, such as improved feature selection or alternative modeling techniques, to achieve meaningful predictions.

Although this implementation follows the correct approach for Bitcoin price prediction by focusing on price movements rather than absolute values, the results indicate poor model performance. The negative R^2 score suggests that the model fails to capture meaningful patterns, and the high MSE and MAE values highlight significant errors in prediction. Despite these shortcomings, this approach aligns with the fundamental economic requirement of predicting price trends rather than raw price values.

TABLE IV
RESULTS OF IMPLEMENTATION 3

Evaluation of Implementation 3	
R2	-0.0036
MSE	76962
MAE	181.93
Epochs	50
Batch Size	64
Training Time (sec)	358.22
Inference Time (sec)	2.59

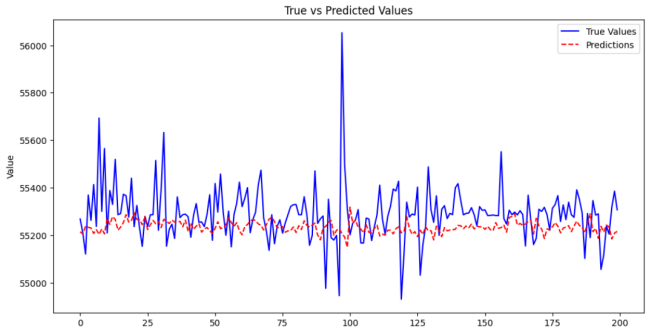

Fig. 14. Plots of prediction and actual values - Implementation 3

The correlation matrix for Implementation 3 reveals key relationships between features and the target variable. While traditional price-related features such as *close*, *open*, *high*, and *low* exhibit strong correlations among themselves, the target variable shows weak or insignificant correlations with most features. This suggests that the model struggles to find meaningful predictive patterns, which aligns with the poor performance observed in the evaluation metrics. Additionally, technical indicators like RSI and moving averages do not show significant correlation with the target, indicating that further feature engineering or alternative transformations may be necessary to improve model effectiveness for future work.

VI. CONCLUSION AND FUTURE WORK

In this project, we explored various methodologies for Bitcoin price prediction, including ARIMA and LSTM-based neural networks. Our results indicate that while traditional statistical models like ARIMA struggle to capture Bitcoin's highly volatile nature, deep learning models, particularly LSTMs, demonstrate superior predictive performance. Among the three implementations, predicting Bitcoin price changes rather than absolute values proved to be a more effective approach.

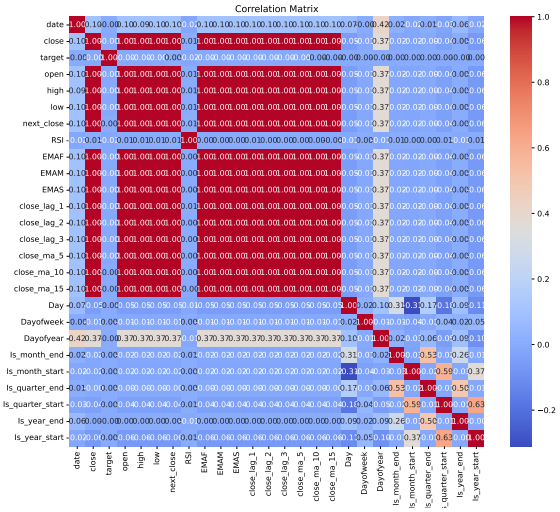

Fig. 15. Correlation matrix for implementation 3

Despite the promising results, challenges remain in achieving consistently high accuracy due to the unpredictable nature of cryptocurrency markets. The correlation analysis suggests that commonly used technical indicators may not always be reliable predictors, highlighting the need for more advanced feature selection and engineering techniques. Additionally, the models tested in this study exhibited varying degrees of effectiveness, with some approaches being theoretically correct but practically limited in their predictive power.

In order to improve predictive accuracy and model robustness, future research could explore the following directions:

- **Hybrid Models:** Combining statistical approaches (such as ARIMA) with deep learning models (LSTM, GRU) to leverage both short-term patterns and long-term dependencies.
- **Feature Engineering:** Incorporating external factors such as trading volume, social media sentiment, macroeconomic indicators, and blockchain analytics to enhance model performance.
- **Reinforcement Learning:** Applying reinforcement learning techniques to optimize trading strategies based on predictive models.
- **Alternative Architectures:** Experimenting with Transformer-based models (e.g., Attention-based models) to capture complex dependencies in financial time series data.
- **Real-time Adaptation:** Implementing online learning mechanisms that allow models to dynamically update and adapt to new market conditions.

Overall, our findings highlight the potential of deep learning for cryptocurrency forecasting while underscoring the need for continuous refinement and innovation in predictive modeling.

References

[1] G. Zhang, B. Eddy Patuwo, and M. Y. Hu, "Forecasting with artificial neural networks:: The state of the art," *International Journal of Forecasting*, vol. 14, no. 1, pp. 35–62, 1998.

[2] CryptoExpert, "Deep learning for bitcoin price prediction." YouTube, n.d. https://youtu.be/5Ym-dOS9ssA?si=dEHWrRtCSQBslMRU.

[3] Data Science at Microsoft, "Introduction to feature engineering for time series forecasting," *Medium*, 2023.

[4] DotData, "Practical guide for feature engineering of time series data," 2023.

[5] R. J. Hyndman and G. Athanasopoulos, *Forecasting: principles and practice (2nd ed.)*. OTexts, 2023.

[6] G. Box and G. Jenkins, *Time Series Analysis: Forecasting and Control*. Holden-Day, 1970.

[7] M. Nagadia, "Bitcoin price prediction using lstm." Kaggle, n.d. https://www.kaggle.com/code/meetnagadia/bitcoin-price-prediction-using-lstm.

[8] L. Programming, "Bitcoin price prediction using lstm." YouTube, n.d. https://www.youtube.com/watch?v=lhrCz6t7rmQ.

[9] C. Versloot, "Build an lstm model with tensorflow and keras," 2023.

[10] F. K. Abdulazeez, "Essential regression evaluation metrics: Mse, rmse, mae, r², and adjusted r²," 2023.

[11] TrainDataHub, "Interpretation of evaluation metrics for regression analysis (mae, mse, rmse, mape, r-squared, and adjusted r-squared)," 2023.