A Comparative Analysis of Machine Learning, Recurrent Neural Networks and Deep Learning Models for Bitcoin Price Forecasting

1st Nguyen Do Duc Minh
IS403.P23
University of Information Technology
22520872@gm.uit.edu.vn

2nd Phan Thanh Cong
IS403.P23
University of Information Technology
22520170@gm.uit.edu.vn

3rd Tran Vu Bao
IS403.P23
University of Information Technology
22520124@gm.uit.edu.vn

4th Phan Thi Thuy Hien
IS403.P23
University of Information Technology
22520423@gm.uit.edu.vn

***Abstract*— *Bitcoin, as the most prominent cryptocurrency, has attracted significant attention from investors, researchers, and financial analysts. However, accurately forecasting Bitcoin prices remains a challenging task due to the market's high volatility and nonlinear dynamics. This paper explores a range of statistical and machine learning models to predict Bitcoin's closing price using historical time-series data from Yahoo Finance (BTC-USD), covering the period from January 1, 2018, to April 29, 2025. Specifically, we evaluate the performance of ARIMA, ARIMAX, XGBoost, LSTM, GRU, and Transformer-based models. These models are assessed using several error metrics, including Root Mean Square Error (RMSE), Mean Absolute Error (MAE), Mean Absolute Percentage Error (MAPE), and the Coefficient of Determination (R^2). Our results provide a comprehensive comparison of traditional and deep learning approaches in the context of Bitcoin price forecasting.*

***Keywords* —** *Bitcoin, Machine Learning, ARIMA, ARIMAX, RNN, XGBoost, LSTM, GRU, Transformer, Yahoo Finance.*

I. INTRODUCTION

Bitcoin, introduced in 2009, has evolved into a global financial asset, widely used both as a speculative investment and a store of value. Unlike traditional assets, Bitcoin operates in a decentralized ecosystem, influenced by a myriad of factors such as investor sentiment, global economic events, and technological developments. Its high volatility and non-stationary behavior make accurate price predictions particularly difficult, yet crucial for investors and financial strategists.

Traditional time-series forecasting methods, such as ARIMA, have been effective for modeling linear dependencies. However, the rapidly changing and nonlinear characteristics of the cryptocurrency market necessitate more advanced modeling techniques. To address this, we examine a blend of statistical and machine learning models, including ARIMA, ARIMAX, XGBoost, LSTM, GRU, and Transformer models.

The primary objective of this study is to predict the next 30 days of Bitcoin closing prices based on historical data spanning more than seven years. By comparing these diverse models, we aim to determine which approaches yield the most accurate short-term forecasts, offering valuable insights for traders, analysts, and financial systems leveraging predictive analytics.

II. RELATED WORK

Numerous studies have attempted to forecast Bitcoin prices using various statistical and computational methods. Traditional models such as ARIMA and ARIMAX have been widely used for their simplicity and interpretability. For example, ARIMA-based forecasting has shown reliable results over short time horizons but often fails to capture nonlinear fluctuations inherent in crypto markets [1].

To improve predictive accuracy, researchers have increasingly turned to machine learning approaches. XGBoost, a powerful ensemble learning method, has been applied to model complex relationships among features beyond just price history [2]. Meanwhile, deep learning models—particularly LSTM and GRU networks—have shown considerable success in handling sequential dependencies and learning from long historical windows [3][4].

More recently, Transformer architectures, known for their success in natural language processing, have been adapted for time-series forecasting. These models leverage attention mechanisms to capture both short-term and long-range dependencies in time-series data, offering advantages in speed and scalability [5].

Despite the breadth of approach, few studies have systematically compared a wide spectrum of traditional, machine learning, and deep learning models in forecasting short-term (e.g., 30-day) Bitcoin price trends. Our work contributes to filling this gap by conducting a comprehensive evaluation of six prominent models on an extensive real-world dataset [6].

III. MATERIALS

A. Data source

The dataset employed in this study is a univariate time series of Bitcoin market data retrieved from Yahoo Finance, covering the period from January 1, 2018, to April 29, 2025. It consists of 2676 observations with seven key attributes: Date, Adjusted Close, Close, High, Low, Open, and Volume. These attributes represent respectively: the trading date, the adjusted closing price reflecting corporate actions such as splits and dividends, the actual closing price at the end of each trading day, the highest and lowest trading prices recorded during the day, the opening price at the market open, and the total volume of Bitcoin traded on that day. This dataset serves as the foundation for training and evaluating predictive models for Bitcoin price forecasting.

B. Data preprocessing

Prior to training the models, the dataset was examined for missing values. The results indicated that there were no missing entries across all records and attributes, ensuring data consistency for subsequent modeling tasks.

Regarding the data splitting strategy, the majority of the models implemented in this study used an 80:20 split ratio, where 80% of the data was allocated for training and the remaining 20% for testing, where the Close price series was partitioned accordingly.

However, for the Transformer-based model, a different strategy was adopted, in which the dataset was divided into training, validation, and test sets. Specifically, the training set comprised 70% of the total data, while both the validation and test sets accounted for 15% each. This multi-stage splitting approach was implemented to support the model's requirement for hyperparameter tuning and to mitigate overfitting.

C. Descriptive statistics

	Adj Close	Close	High
Count	2676	2676	2676
Mean	30911.6111	30911.6111	31555.7995
Std	25349.6460	25349.6460	25855.9370
Min	3236.7617	3236.7617	3275.3779
25%	9192.3413	9192.3413	9339.0876
50%	23651.3789	23651.3789	24124.5283
75%	46404.7441	46404.7441	47402.9258
Max	106146.2656	106146.2656	109114.8828

Table 1. Descriptive Statistics

	Low	Open	Volume
Count	2676	2676	2.676000e+03
Mean	30174.3656	30883.2541	2.763285e+10
Std	24768.7148	25323.6474	2.004455e+10
Min	3191.3035	3236.2747	2.923670e+09
25%	9013.5144	9186.8108	1.436387e+10
50%	23162.1289	23634.1514	2.445092e+10
75%	45128.0605	46365.2607	3.615642e+10
Max	105291.7344	106147.296	3.509679e+11

Table 2. Descriptive Statistics

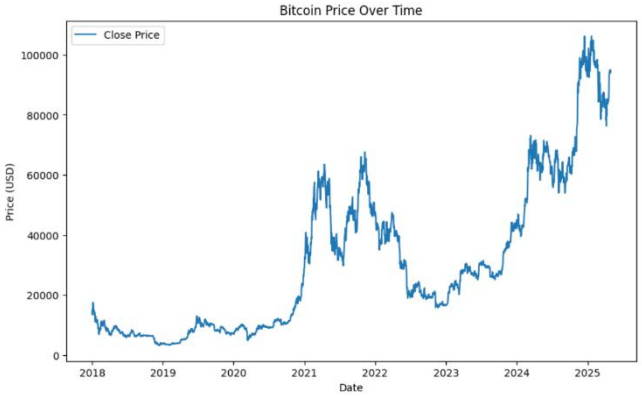

Figure 1. Bitcoin Close Price Over Time

IV. METHODOLOGY

A. ARIMA

ARIMA stands for Auto Regressive Integrated Moving Average.

The basic model in the time series analysis is the ARIMA model. It is a combination of two processes – *autoregressive* (AR) and *moving average* (MA), which is weighted delayed random components. The letter (I) in the model's name indicates the level of integration of the analyzed variable. Integrated variables are variables that can become stationary through differentiation. The structure of ARIMA is based on the phenomenon of autocorrelation. ARIMA can be used for modeling stationary time series or non-stationary time series that can become stationary through differentiation. [7].

There are seasonal and non-seasonal ARIMA models that can be used for forecasting:

- Non-Seasonal ARIMA model.
- Seasonal ARIMA (SARIMA) models.

Non-Seasonal ARIMA(p, d, q):

- p: Periods to lag for e.g: (if p = 3 then we will use the three previous periods of our time series in the autoregressive portion of the calculation), *p* helps adjust the line that is being fitted to forecast the series.
- q: This variable denotes the lag of the error component, where error component is a part of the time series not explained by trend or seasonality.
- d: In an ARIMA model we transform a time series into stationary one (series without trend or seasonality) using differencing. Parameter *d* refers to the number of different transformations required by the time series to get stationery.

Stationary time series is when the mean and variance are constant over time. It is easier to predict when the series is stationary. Differencing is a method of transforming a non-stationary time series into a stationary one. This is an important step in preparing data to be used in an ARIMA model.

In the experimental phase, the Augmented Dickey-Fuller (ADF) test was first conducted to examine the stationarity of the original time series. The test yielded a p-value of 0.917, which exceeds the conventional threshold of 0.05, indicating that the series is non-stationary. Consequently, first-order differencing was applied to the data, followed by a second ADF test. The result showed a significantly lower p-value of 1.3845e-13, which is well below 0.05, confirming that the different series is now stationary. This implies that the series satisfies the stationarity condition required for ARIMA modeling, and the differencing order $d = 1$ was thereby determined.

Following this, the Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) plots were examined to identify appropriate values for the p and q parameters (as shown in *Figure 2*). Both plots demonstrated a sharp drop after lag 1, with no significant spikes, thereafter, suggesting that the series becomes approximately white noise after differencing. Based on this observation, the **ARIMA(1,1,1)** model was selected as a suitable configuration for capturing the underlying patterns in the data.

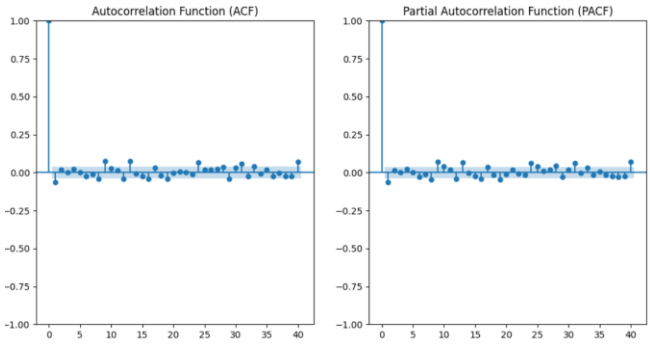

Figure 2. ACF and PACF used to identify ARIMA model parameters

B. ARIMAX

While the ARIMA model relies solely on the past values and errors of the target variable, the ARIMAX (AutoRegressive Integrated Moving Average with Exogenous Variables) model extends this framework by incorporating one or more external (exogenous) variables that are believed to influence the target time series. This enhancement enables ARIMAX to account for external factors, thereby potentially improving the model's predictive capability.

In this study, the variable *Volume* - representing the trading volume of Bitcoin - was selected as the exogenous input for the ARIMAX model. The decision was based on the observed Pearson correlation coefficient between the target variable Close (closing price) and Volume, which was *0.4943*. This value suggests a moderate correlation: not too weak to be negligible, and not too strong to indicate redundancy. Thus, Volume exhibits a measurable influence on the Close price and is considered a reasonable candidate for inclusion as an exogenous variable.

The experimental procedure for the ARIMAX model followed a similar approach to that of the ARIMA model. Since the stationarity of the Close series was already established through the ADF test and first-order differencing (resulting in differencing order $d = 1$), no additional preprocessing was required. The differenced series served as the basis for identifying the autoregressive (p) and moving average (q) parameters using the ACF and PACF plots. Both plots exhibited a sharp decline after lag 1, with no significant spikes beyond that point, indicating a structure amenable to a low-order ARIMAX configuration.

Based on this analysis, the **ARIMAX(1,1,0)** model was chosen as the optimal configuration for capturing the dynamics of Bitcoin's closing price while accounting for the influence of trading volume.

C. XGBoost

XGBoost *(eXtreme Gradient Boosting)* is an optimized gradient boosting framework introduced by Tianqi Chen to enhance speed and performance in machine learning tasks. It belongs to the family of ensemble methods, where multiple weak learners, typically decision trees, are combined sequentially to produce a strong predictive model. XGBoost is well-known for its scalability, high accuracy, and efficiency, particularly on structured data [8].

The core idea of XGBoost lies in minimizing a regularized objective function that balances training loss and model complexity. The general form of the objective function is defined as:

$$L(\phi) = \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t)}\right) + \sum_{k=1}^{t} \Omega(f_k)$$

Where:

- $l\left(y_i, \hat{y}_i^{(t)}\right)$ is a differentiable convex loss function (e.g., squared error) that measures the difference between the true value y_i and the predicted value $\hat{y}_i^{(t)}$
- $\Omega(f_k)$ is a regularization term that penalizes the complexity of the model.

The regularization term $\Omega(f)$ for each regression tree is defined as:

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

Where:

- T is the number of leaves in the tree.
- w_j^2 is the score on leaf j
- γ and λ are regularization parameters.

To determine the optimal structure of decision trees, XGBoost computes the gain of each split using first- and

second-order derivatives (gradient and Hessian) of the loss function. The gain is calculated as:

$$Gain = \frac{1}{2}\left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda}\right] - \gamma$$

Where:

- G_L, H_L : sum of first and second derivatives (gradient and hessian) for the left child.
- G_R, H_R sum.
- of gradient and hessian for the right child.

In addition to gradient boosting, XGBoost introduces several optimization techniques, including tree pruning, column subsampling, and parallelized tree construction, which significantly improve computational speed and predictive performance.

Although not originally developed for time series problems, XGBoost can be effectively applied to time series forecasting by converting the sequential data into a supervised learning problem. This is achieved by engineering lag features (e.g., past closing prices), rolling statistics (e.g., moving averages, volatility), and temporal features (e.g., day of the week, month).

In this study, XGBoost is employed to forecast Bitcoin closing prices for the next 30 days. Historical data from Yahoo Finance (BTC-USD) is used to construct input features, including lagged prices and temporal indicators. The model learns the complex nonlinear relationships in the data and predicts future values based on past trends and patterns.

XGBoost is particularly effective for financial forecasting tasks due to its ability to handle non-stationary inputs, incorporate exogenous variables, and control overfitting through built-in regularization mechanisms.

D. RNN

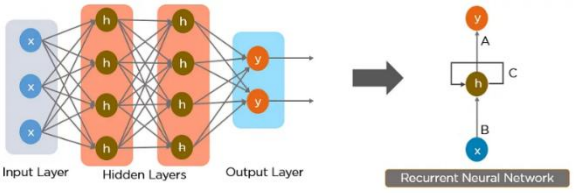

Figure 3. RNN Architecture

Recurrent Neural Networks is a more flexible model, since it encodes the temporal context in its feedback connections, which can capture the time varying dynamics of the underlying system RNNs are learning machines that recursively compute new states by applying transfer functions to previous states and inputs. Typical transfer functions are composed of an affine transformation followed by a nonlinear function, which are chosen depending on the nature of the problem at hand. [9]

An activation function determines whether a neuron should be activated. The nonlinear functions typically convert the output of a given neuron to a value between 0 and 1 or -1 and 1.

E. LSTM

RNN is a time-series neural network. The interconnection structure between the hidden layers reflects the interaction between time series. However, there are vital problems that exist in RNN: the fast gradient descent problem and nonconvergent problem. Fortunately, the bi-directional LSTM model can solve the gradient problem of RNN network by adding gates and using the context relation of forward and backward time directions in time series, improving the prediction accuracy subsequently.

LSTM had achieved surprising performance in the NLP field. LSTM aims to resolve long-term dependence problems based on improved RNN (Annotation) neural network. Keeping information in mind for a long time is an inherent characteristic of LSTM. All RNN models have a chain form of repetitive neural network modules.

Similarly, as a variant of RNN, LSTM also has this chain module structure, but with different repetitive modules and layers. LSTM has three more gates than RNN with only the tanh layer.

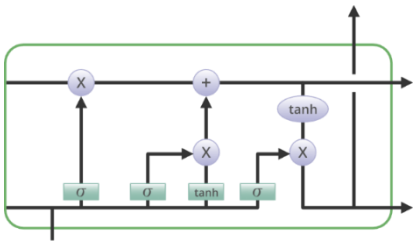

Figure 4. LSTM Architecture

LSTM equations:

$$\text{Forget Gate: } f_t = \sigma(X_t U_f + H_{t-1} W_f)$$
$$\text{Input Gate: } i_t = \sigma(X_t U_i + H_{t-1} W_i)$$
$$\text{Cell Gate: } c_t = tanh(W_c X_t + U_c H_{t-1})$$
$$\text{Output Gate: } o_t = \Phi h(W_o X_t + U_0 H_{t-1})$$
$$\text{Cell State: } c_t = f_t c_{t-1} + i_t c_t$$

Bidirectional long-short term memory (Bi-LSTM) is a technique that allows any neural network to store sequence information both forward and backward. Bi-LSTM allows input flow in both directions, whereas normal LSTM only allows input flow in one direction

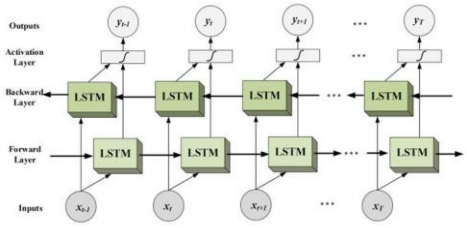

Figure 5. Bidirectional LSTM

The Long Short-Term Memory (LSTM) model was utilized in this study to predict Bitcoin closing prices using historical data from Yahoo Finance (January 1, 2018, to April 29, 2025). Implemented via TensorFlow Keras, the LSTM addresses long-term dependencies and mitigates vanishing gradient issues.

The model utilized in the current study features a sequential architecture with three LSTM layers, each with 50 units: the first layer (return_sequences=True, input shape (time_step, 1)) is followed by a Dropout layer (rate 0.2), the second layer also uses 50 units with return_sequences=True and Dropout (0.2), and the third layer (50 units, no sequence return) concludes with a Dropout (0.2) and a Dense layer (1 unit), compiled with the Adam optimizer and MSE loss, totaling 38,601 trainable parameters with output shapes (None, 15, 50) (10,400 params), (None, 15, 50) (20,200 params), (None, 50) (20,200 params), and (None, 1) (51 params).

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 15, 50)	10,400
dropout (Dropout)	(None, 15, 50)	0
lstm_1 (LSTM)	(None, 15, 50)	20,200
dropout_1 (Dropout)	(None, 15, 50)	0
lstm_2 (LSTM)	(None, 50)	20,200
dropout_2 (Dropout)	(None, 50)	0
dense (Dense)	(None, 1)	51

Figure 6. LSTM model layers

The three-layer, 50-unit LSTM with dropout was chosen to model non-linear trends, with the 15-day step aligning with short-term market patterns. Overfitting suggests a need for regularization (e.g., higher dropout or early stopping) to improve generalization for 30-day forecasts (96,205.18–98,225.40 USD).

We trained the model for 100 epochs (batch size 32). Overall, the achieved a training RMSE of 2,991.90 USD (R^2 0.968) but a test RMSE of 3,530.29 USD (R^2 0.299), indicating the presence of overfitting issue.

F. GRU

A Gated Recurrent Unit (GRU) is a variant of the RNN architecture and uses gating mechanisms to control and manage the flow of information between cells in the neural network. GRUs can be considered a relatively new architecture, especially when compared to the widely adopted LSTM.

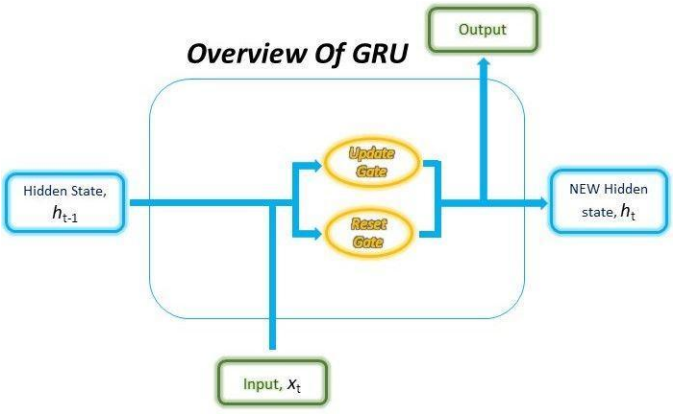

Figure 7. Overview of GRU

The structure of the GRU allows it to adaptively capture dependencies from large sequences of data without discarding information from earlier parts of the sequence. This solve the vanishing/exploding gradient problem of traditional RNNs.

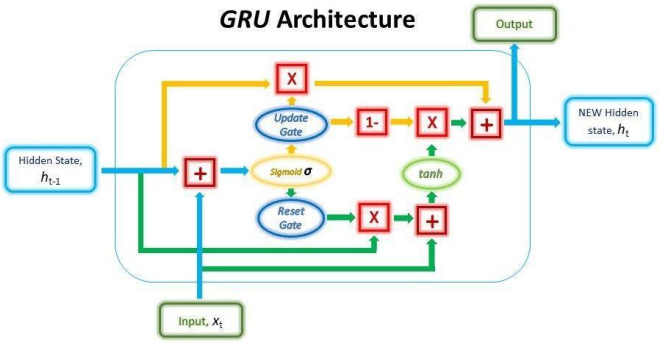

Figure 8. GRU architecture

The ability of the GRU to hold on to long-term dependencies or memory stems from the computations within the GRU cell to produce the hidden state. While LSTMs have two different states passed between the cells — the cell state and hidden state, which carry the long and short-term memory, respectively — GRUs only have one hidden state transferred between time steps. This hidden state is able to hold both the long-term and short-term dependencies at the same time due to the gating mechanisms and computations that the hidden state and input data go through.

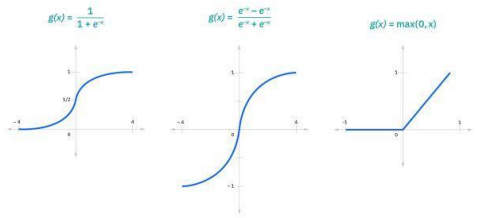

Figure 9. GRU activation function

GRU equations:

Reset Gate:

$$gate_{reset} = \sigma(W_{input_{reset}}.x_t + W_{hidden_{reset}}.h_{t-1})$$

$$r = tanh\ tanh\ (gate_{reset} \odot W_{h_1}.h_{t-1}) + W_{x_1}.x_t)$$

Update Gate:

$$gate_{update} = \sigma(W_{input_{update}}.x_t + W_{hidden_{update}}.h_{t-1})$$

$$u = gate_{reset} \odot h_{t-1}$$

Combining the outputs:

$$h_t = r \odot (1 - gate_{update}) + u$$

Given the capabilities of the Gated Recurrent Unit (GRU) model, an advanced variant of Recurrent Neural Networks (RNNs) with simpler architecture compared to the Long Short-Term Memory model (LSTM), this study implemented GRU to predict Bitcoin closing prices using historical data from Yahoo

Finance (January 1, 2018, to April 29, 2025), leveraging the TensorFlow Keras framework to capture temporal dependencies effectively [16]. The model architecture comprises three GRU layers, each with 50 units: the first layer (return_sequences=True, input shape (time_step, 1)) is followed by a Dropout layer (rate 0.2), the second layer also uses 50 units with return_sequences=True and Dropout (0.2), and the third layer (50 units, no sequence return) concludes with a Dropout (0.2) and a Dense layer (1 unit), compiled with the Adam optimizer and MSE loss, totaling 38,601 trainable parameters with output shapes (None, 15, 50) (10,400 params), (None, 15, 50) (20,200 params), (None, 50) (20,200 params), and (None, 1) (51 params)

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 15, 50)	10,400
dropout (Dropout)	(None, 15, 50)	0
lstm_1 (LSTM)	(None, 15, 50)	20,200
dropout_1 (Dropout)	(None, 15, 50)	0
lstm_2 (LSTM)	(None, 50)	20,200
dropout_2 (Dropout)	(None, 50)	0
dense (Dense)	(None, 1)	51

Figure 10. GRU model layers

The GRU's three-layer structure with dropout was selected to model non-linear trends, validated by its ability to forecast 30-day prices (e.g., $90,329.16 to $76,088.75). Training was conducted using a 15-day time step, creating input shapes tailored to short-term volatility patterns, and spanned 100 epochs with a batch size of 32, incorporating callbacks such as EarlyStopping and ReduceLROnPlateau to optimize performance. This resulted in a training RMSE of $1,169.53 ($R^2$ 0.9946) and a test RMSE of $3,354.65 ($R^2$ 0.9686), indicating robust performance with low overfitting (R^2 difference 0.0261). However, recommendations include regular model updates and cautious application alongside technical indicators for investment decisions. [10].

G. Transformer

For years, Recurrent Neural Networks (RNNs) and their more sophisticated variants like LSTMs (Long Short-Term Memory) and GRUs (Gated Recurrent Units) were the dominant architectures for processing sequential data. Their ability to maintain a "memory" of past inputs made them well-suited for tasks like natural language processing, speech recognition, and time series analysis. However, RNNs faced significant limitations. They struggled with long-range dependencies, where information from earlier parts of a sequence became difficult to access and utilize effectively as the sequence grew longer. This was largely due to the vanishing and exploding gradient problems during training, hindering the learning of connections across distant elements. Furthermore, the inherent sequential processing of RNNs made them difficult to parallelize, leading to longer training times, especially on increasingly large datasets. The need for models that could efficiently handle long sequences, capture global context, and leverage parallel computation paved the way for the emergence of Transformer architecture, a paradigm shift that revolutionized the field [11].

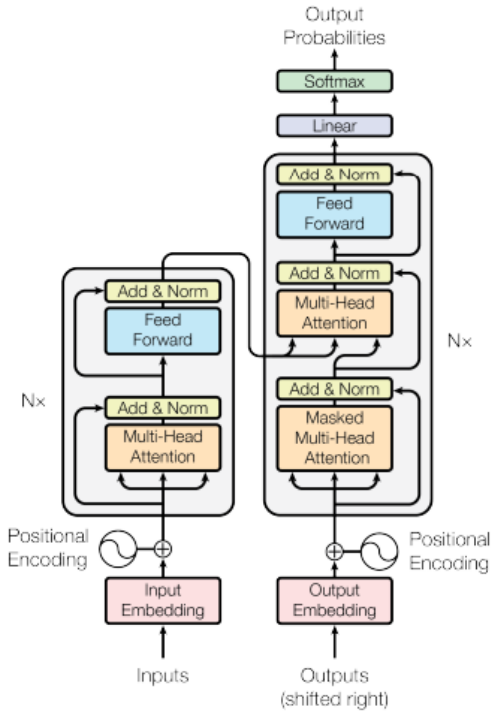

Figure 11. The Transformer model architecture

Encoder and Decoder Stacks

- **Encoder:** The encoder is composed of a stack of $N = 6$ identical layers. Each layer has two sub-layers. The first is a multi-head self-attention mechanism, and the second is a simple, position-wise fully connected feed-forward network. The output of each sub-layer is $\mathrm{LayerNorm}(x + \mathrm{Sublayer}(x))$ where $\mathrm{Sublayer}(x)$ is the function implemented by the sub-layer itself. To facilitate these residual connections, all sub-layers in the model, as well as the embedding layers, produce outputs of dimension $d_{model} = 512$.

- **Decoder:** The decoder is also composed of a stack of $N = 6$ identical layers. In addition to the two sub-layers in each encoder layer, the decoder inserts a third sub-layer, which performs multi-head attention over the output of the encoder stack. Like the encoder, employing residual connections around each of the sub-layers, followed by layer normalization. Also modifying the self-attention sub-layer in the decoder stack to prevent positions from attending to subsequent positions. This masking, combined with fact that the output embeddings are offset by one position, ensures that the predictions for position i can depend only on the known outputs at positions less than i.

Attention Mechanism

The attention mechanism is a fundamental component in the Transformer architecture, enabling the model to dynamically focus on relevant parts of the input sequence when generating representations. Formally, an attention function can be described as a mapping from a query and a set of key-value pairs to an output, where the query (Q), keys (K), values (V), and output are all vectors. The output is computed as a weighted sum of the values, with the weight assigned to each value being determined by a compatibility function between the query and the corresponding key. This mechanism allows the model to capture dependencies regardless of their distance in the input or output sequences, a key advantage over traditional recurrent or convolutional models.

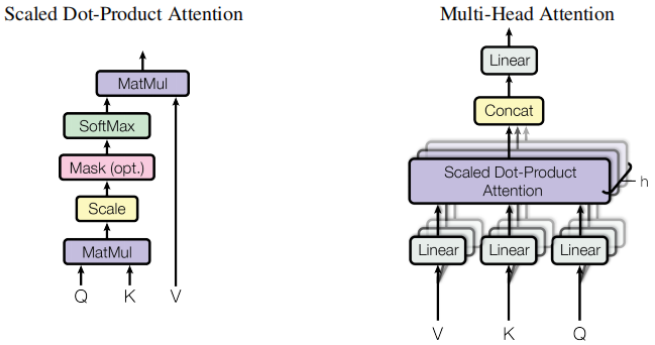

Figure 12. Scaled Dot-Product Attention (left) and Multi-Head Attention consists of several attention layers running in parallel

- **Scaled Dot-Product Attention**

  Given matrices of queries $Q \in \mathbb{R}^{n_q \times d_k}$, keys $K \in \mathbb{R}^{n_k \times d_k}$, and values $V \in \mathbb{R}^{n_v \times d_v}$, the attention function is computed as:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

  Here, d_k denotes the dimensionality of the key vectors. The dot products QK^T measure the similarity between queries and keys. The result is scaled by $\sqrt{d_k}$ to mitigate the issue of large variance in dot product values for high-dimensional vectors, which can result in vanishing gradients after the softmax operation.
  The **SoftMax** function then transforms these scaled scores into probability distribution, highlighting the most relevant positions in the sequence. Finally, the resulting weights are used to compute a weighted sum of the value vectors.

- **Multi-Head Attention**

  The multi-head attention mechanism is a key innovation in the Transformer architecture that enables the model to capture different types of relationships and dependencies within sequences by attending to information from multiple representation subspaces simultaneously.
  Instead of performing a single attention function with queries, keys, and values, multi-head attention runs multiple self-attention operations in parallel, each referred to as an attention head. Each head uses distinct linear projections of the input to learn from different representation subspaces. The outputs of these attention heads are then concatenated and linearly transformed to produce the final output.
  Formally, the multi-head attention is defined as:

$$MultiHead(Q, K, V) = Concat(head_1 \ldots, head_h)W^O$$

  where each attention head is computed as:

$$head_i = Attention\,(QW_i^Q, KW_i^K, VW_i^V)$$

  Here:

  - $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$
  - $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$
  - $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$
    are learnable parameter matrices for the i-th head that project the inputs into query, key, and value subspaces, respectively.
  - $W^O \in \mathbb{R}^{hd_v \times d_{model}}$ is the output projection matrix that transforms the concatenated output back to the model dimension.

  This mechanism allows each head to focus on different positions and types of dependencies in the input sequence, improving the model's expressiveness. Additionally, by distributing the attention mechanism across multiple heads, the model can capture a richer set of patterns while maintaining computational efficiency.

Masking in Decoder Self-Attention

In sequence generation tasks, the decoder must not access future tokens during training, to ensure that predictions for position i depend only on positions less than or equal to i. To enforce this autoregressive property, the Transformer applies a **look-ahead mask** (also called a causal mask) to the self-attention mechanism within the decoder.
This masking works by setting attention weights corresponding to future positions to negative infinity before the SoftMax operation. Formally, if Q, K, and V are the query, key, and value matrices, and M is a binary mask matrix with:

$$M_{i,j} = \begin{cases} 0 & if\ j \le i \\ -\infty & if\ j > i \end{cases}$$

then the masked attention is computed as:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}} + M\right)V$$

This ensures that each token only attends to the current and previous positions in the sequence during training.

Position-wise Feed-Forward Networks

In addition to the attention mechanisms, each layer in both the encoder and decoder includes a position-wise feed-forward network (FFN). This sub-layer is applied independently to each position in the sequence and consists of two linear transformations separated by a non-linear activation function:

$$FFN(x) = \max(0, xW_1 + b_1)\,W_2 + b_2$$

Here $W_1 \in \mathbb{R}^{d_{model} \times d_{ff}}$, $W_2 \in \mathbb{R}^{d_{ff} \times d_{model}}$ and b_1, b_2 are the bias terms. In the original implementation, d_{model} = 512 and

d_{ff} = 2048. Although the same feed-forward network is applied to all positions within a layer, the parameters W_1, W_2 are unique to each layer, allowing hierarchical abstraction across depth. This operation can also be interpreted as a 1D convolution with kernel size 1, which enables position-wise computation with shared parameters across tokens.

Embeddings and Output SoftMax

To handle discrete input tokens, the model employs learned embedding layers that map input and output tokens to continuous vectors of dimension d_{model}. For the output layer, a learned linear transformation followed by a softmax activation is used to project the decoder's output into a probability distribution over the vocabulary, enabling prediction of the next token in the sequence. This mechanism aligns with standard practices in sequence-to-sequence modeling.

Positional Encoding

Since the Transformer model lacks any recurrence or convolution, it does not inherently model the order of tokens. To address this, positional encodings are added to the input embeddings at the bottom of the encoder and decoder stacks. These encodings carry information about the absolute or relative position of each token in the sequence.
The positional encodings share the same dimensionality as the token embeddings, d_{model}, which enables their element-wise summation. Two main types of positional encodings are commonly used: fixed (e.g., sinusoidal) and learned. In the original Transformer, sinusoidal functions of varying frequencies are used to encode positions, providing a fixed but continuous and generalizable representation of position.

Residual Connections and Layer Normalization

Each sub-layer in both the encoder and decoder is wrapped in a residual connection, followed by layer normalization. This is expressed as:

$$LayerOutput = LayerNorm(x + Sublayer(x))$$

This structure helps stabilize training by mitigating vanishing or exploding gradients and allows gradients to propagate more easily through deep networks. Layer normalization normalizes across the features of each token embedding (as opposed to across a batch, as in batch normalization), which is more suitable for varying sequence lengths.

Model Architecture and Configuration for Bitcoin Price Prediction

This study implements a novel Transformer-based architecture specifically designed for cryptocurrency price forecasting, incorporating Time2Vector embedding and asymmetric encoder-decoder structures optimized for financial time series prediction.

1. Core Architecture Parameters

The proposed model employs a carefully designed configuration that balances computational efficiency with predictive accuracy. The core architectural parameters are summarized in Table 3.

Table 3. Core Transformer Architecture Configuration

Parameter	Value	Rationale
Model Dimension (d_model)	128	Sufficient capacity for feature representation while maintaining computational efficiency
Attention Heads (nhead)	8	Multi-scale attention mechanism enabling focus on different temporal patterns
Encoder Layers	6	Deep encoding for comprehensive input pattern recognition
Decoder Layers	3	Asymmetric design emphasizing input understanding over output generation
Feedforward Dimension	512	4 × expansion ratio following standard Transformer practices
Dropout Rate	0.1	Regularization to prevent overfitting on financial data
Activation Function	GELU	Superior performance compared to ReLU in Transformer architectures
Model Dimension (d_model)	128	Sufficient capacity for feature representation while maintaining computational efficiency
Attention Heads (nhead)	8	Multi-scale attention mechanism enabling focus on different temporal patterns

2. Temporal Sequence Configuration

The model processes financial time series data using a sliding window approach with the following specifications:

- Input Sequence Length (L_src): 15 trading days.
- Prediction Horizon (L_tgt): 5 trading days.
- Sequence Overlap: 1 day for data augmentation.
- Feature Dimensionality: Variable based on technical indicators.

The 15-day input window was selected to capture short to medium-term market dynamics while avoiding the noise associated with longer historical sequences. The 5-day prediction horizon provides a practical balance between forecast accuracy and utility for trading applications.

3. Time2Vector Embedding Layer

A critical innovation in this architecture is the integration of Time2Vector embedding, which enhances the model's ability to capture temporal patterns inherent in financial data:

$$Time2Vector(x) = [W_0 x + b_0, \sin(W_1 x + b_1), \ldots, \sin(W_k x + b_k), x]$$

Where:

- Linear Features: 8 components capturing trend information
- Periodic Features: 8 sine-based components for cyclical patterns
- Original Features: Preserved input dimensions

This embedding layer addresses the limitation of standard positional encoding in capturing both linear trends and cyclical behaviors common in cryptocurrency markets.

V. RESULT

A. Performance measure metrics

To evaluate the forecasting performance of the proposed models, four widely used error metrics were employed: RMSE, MAE, MAPE, and R^2. These metrics provide complementary insights into the accuracy and reliability of the predicted values.

➢ *Root Mean Square Error (RMSE)*

The RMSE measures the square root of the average squared differences between the actual and predicted values. It penalizes larger errors more heavily and is sensitive to outliers. Lower RMSE values indicate better model performance.

$$RMSE = \sqrt{\frac{1}{n}\sum_{t=1}^{n}(y_t - \hat{y}_t)^2}$$

Where:

- y_t: Actual value at time step t
- $\hat{y}_t$: Predicted value at time step t
- n: Total number of observations.

➢ *Mean Absolute Error (MAE)*

The MAE quantifies the average magnitude of the absolute errors between actual and predicted values. Unlike RMSE, it treats all deviations equally and is more robust to outliers.

$$MAE = \frac{1}{n}\sum_{t=1}^{n}|y_t - \hat{y}_t|$$

Where:

- $y_t, \hat{y}_t$ and n: Defined as above.

➢ *Mean Absolute Percentage Error (MAPE)*

The MAPE represents the average absolute percent error between actual and predicted values, expressed as a percentage. It provides a scale-independent measure but can be sensitive when actual values are close to zero.

$$MAPE = \frac{100\%}{n}\sum_{t=1}^{n}\left|\frac{y_t - \hat{y}_t}{y_t}\right|$$

Where:

- y_t: Actual value at time step t (must be non-zero)
- $\hat{y}_t$: Predicted value at time step t
- n: Total number of observations.

➢ *Coefficient of Determination (R^2 Score)*

The R^2 score measures the proportion of the variance in the dependent variable that is predictable from the independent variables. It ranges from 0 to 1, where values closer to 1 indicate better model performance.

$$R^2 = 1 - \frac{\sum_{t=1}^{n}(y_t - \hat{y}_t)^2}{\sum_{t=1}^{n}(y_t - \bar{y})^2}$$

Where:

- y_t: Actual value at time step t
- $\hat{y}_t$: Predicted value at time step t
- $\bar{y}$: Mean of actual values.
- n: Total number of observations.

B. Predicting on test set

- ARIMA

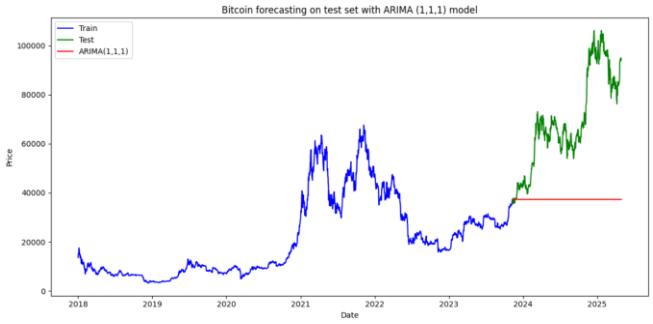

Figure 13. Prediction on test set using ARIMA(1,1,1)

- ARIMAX

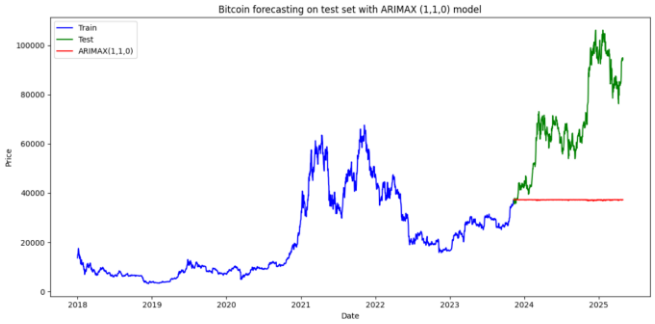

Figure 14. Prediction on test set using ARIMAX(1,1,0)

- XGBoost

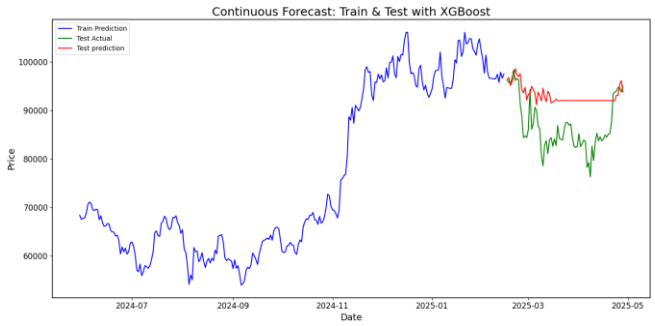

Figure 15. Prediction on test set using XGBoost

- LSTM

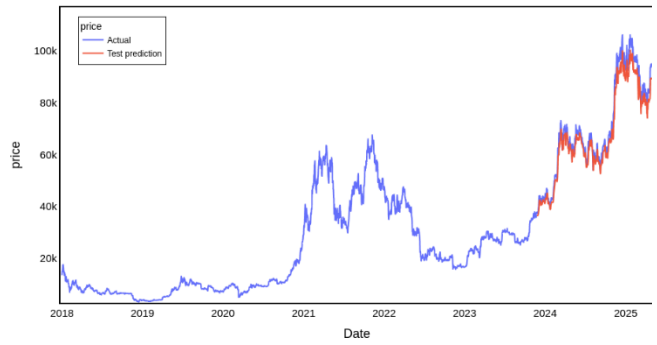

Figure 16. Prediction on test set using LSTM

- GRU

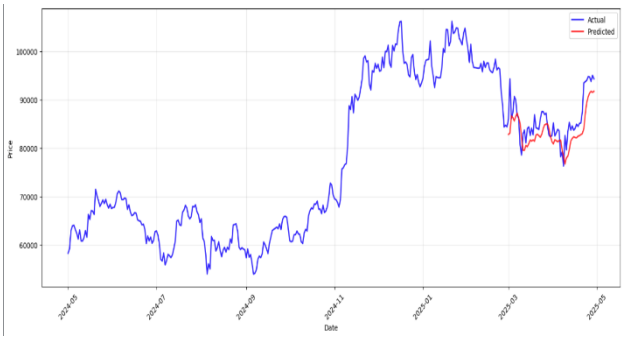

Figure 17. Prediction on test set using GRU

- Transformer

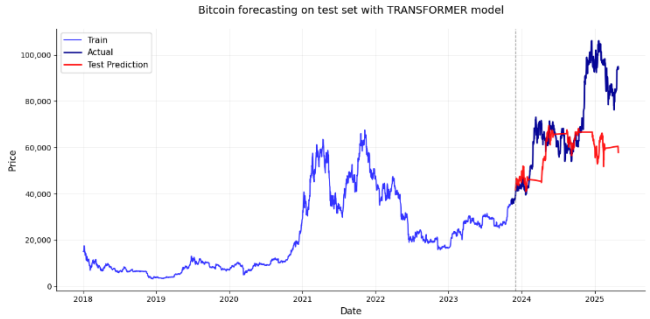

Figure 18. Prediction on test set using Transformer

C. Predicting Bitcoin price for the next 30 days

- ARIMA

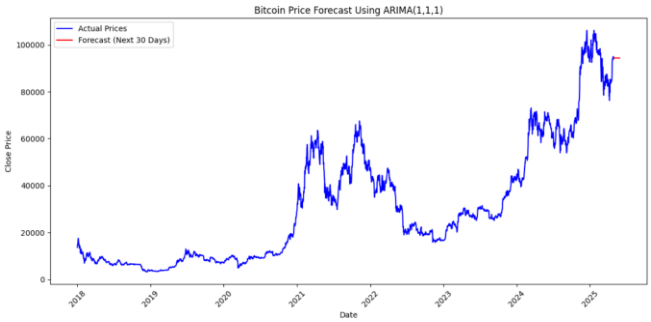

Figure 19. Predict next 30 days using ARIMA(1,1,1)

- ARIMAX

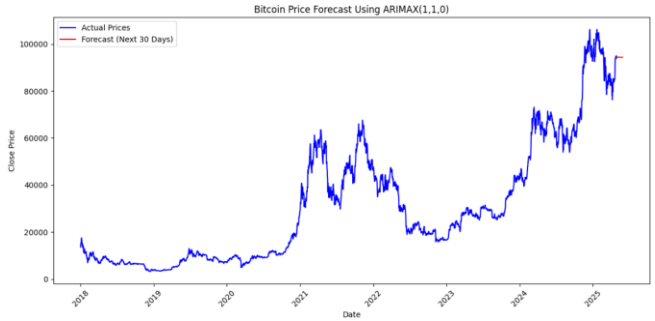

Figure 20. Predict next 30 days using ARIMAX(1,1,0)

- XGBoost

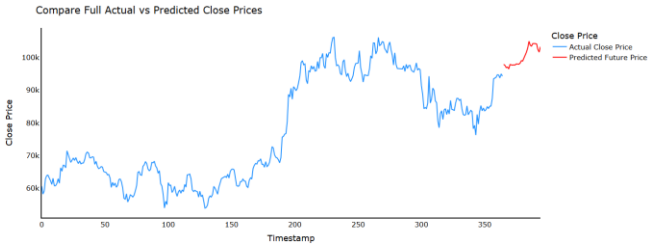

Figure 21. Predict next 30 days using XGBoost

- LSTM

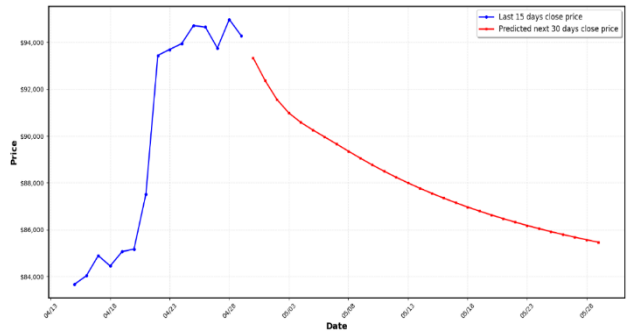

Figure 22. Predict next 30 days using LSTM

- GRU

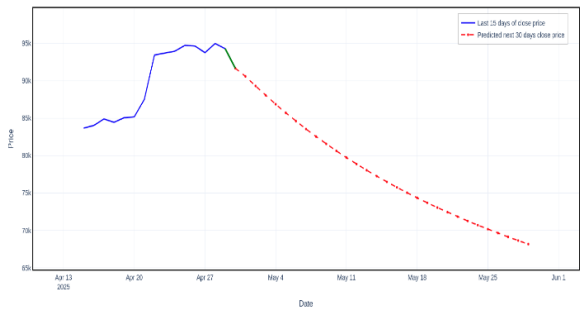

Figure 23. Predict next 30 days using GRU

- Transformer

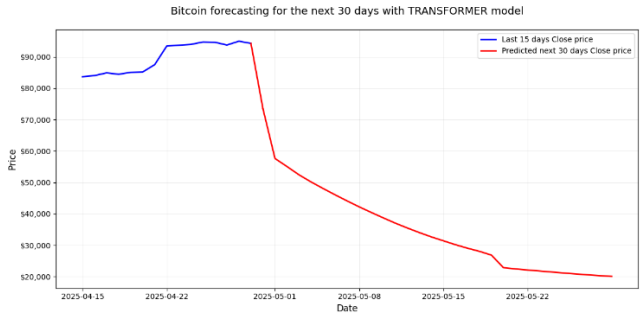

Figure 24. Predict next 30 days using Transformer

D. Model evaluation and comparative analysis

This section evaluates the predictive performance of six models - ARIMA, ARIMAX, XGBoost, LSTM, GRU, and Transformer - in forecasting Bitcoin closing prices on an unseen test set. The evaluation is based on four performance metrics: Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), Mean Absolute Percentage Error (MAPE), and the Coefficient of Determination (R^2). These metrics assess the accuracy, error magnitude, and explanatory power of each model. Additionally, visual representations of the models' predictions against actual test data are analyzed to provide further insight into their performance.

The performance metrics for each model are presented in Table 4, calculated on the test set to ensure an unbiased assessment of predictive capability.

Model/Metrics	RMSE	MAE	MAPE	R^2
ARIMA	37,113.26	31,951.77	41.68%	-2.845
ARIMAX	37,194.02	32,019.16	41.76%	-2.86
XGBoost	21,477.11	15,641.12	17.65%	-0.97
LSTM	2,119.65	1,558.96	2.25%	0.9867
GRU	2,808.44	2,085.24	2.85%	0.978
Transformer	15,798.00	11,017.00	13.16%	0.25

Table 4. Performance measures between models on test set

The results reveal substantial differences in model performance. The LSTM and GRU models exhibit the lowest error metrics and highest R^2 values, indicating superior predictive accuracy. Conversely, ARIMA and ARIMAX show the highest errors and negative R^2 values, suggesting poor fit. XGBoost and Transformer models fall between these extremes, with XGBoost performing poorly and Transformer showing moderate capability.

The ARIMA and ARIMAX models exhibit the weakest performance, with RMSE values exceeding 37,000, MAE values above 31,900, and MAPE values around 41.7%. Their negative R^2 values (-2.845 and -2.86, respectively) indicate that they fail to explain the variance in Bitcoin prices, performing worse than a simple mean-based model. The visual representations (Figures 13 and 14) corroborate these findings, showing flat prediction lines at approximately 40,000 and 30,000, respectively, while the

actual test data rises sharply to 90,000 by late 2024. This inability to capture dynamic trends highlights the limitations of these linear statistical models in modeling the non-linear, volatile nature of Bitcoin prices.

The XGBoost model demonstrates moderate improvement over ARIMA and ARIMAX, with an RMSE of 21,477.11, MAE of 15,641.12, and MAPE of 17.65%. However, its R^2 of -0.97 remains negative, indicating a poor fit to the test data. Figure 15 shows the model's predictions as a nearly flat line around 80,000 from January 2025 to May 2025, failing to reflect the actual price fluctuations between 70,000 and 80,000. This suggests that, despite its capacity for non-linear modeling, XGBoost struggles with the sequential dependencies of time series data, resulting in limited predictive accuracy.

The LSTM and GRU models outperform all others, with RMSE values of 2,119.65 and 2,808.44, MAE values of 1,558.96 and 2,085.24, and MAPE values of 2.25% and 2.85%, respectively. Their R^2 scores of 0.9867 and 0.978 indicate that they explain over 97% of the variance in the test data. Visually, Figures 16 and 17 demonstrate that the predicted prices closely align with the actual prices. For LSTM, the predictions track the actual data from 2018 to 2025, capturing peaks at 60,000 and 90,000. For GRU, the predictions from January 2025 to May 2025 mirror the actual decline and recovery, with minor deviations. These results underscore the effectiveness of these recurrent neural networks in modeling complex sequential patterns in Bitcoin price data.

The Transformer model shows moderate performance, with an RMSE of 15,798.00, MAE of 11,017.00, and MAPE of 13.16%. Its R^2 of 0.25 suggests it explains only a quarter of the variance in the test data. Figure 18 reveals that while the predictions follow the general trend from mid-2022 to 2023, they significantly underestimate the peak at 60,000 in 2024, hovering around 40,000. This indicates that the Transformer model captures some patterns but struggles with extreme volatility, potentially requiring further optimization.

VI. CONCLUSION

This study presents a comprehensive comparative analysis of traditional statistical models (ARIMA, ARIMAX), machine learning approaches (XGBoost), and deep learning architectures (RNN, LSTM, GRU, Transformer) for short-term Bitcoin price forecasting. Utilizing historical BTC-USD data from Yahoo Finance spanning over seven years, we implemented and evaluated each model based on standard forecasting metrics, including RMSE, MAE, MAPE, and R^2.

The experimental results indicate that while ARIMA and ARIMAX models offer interpretability and are effective for capturing linear trends, they underperform in scenarios characterized by high volatility and nonlinear dependencies—features inherent to cryptocurrency markets. XGBoost, with its ensemble learning framework, demonstrated improved performance but relied heavily on feature engineering to capture temporal relationships.

Deep learning models, particularly GRU and Transformer, achieved superior predictive accuracy. The GRU model yielded the best balance between training and test performance, indicating strong generalization and reduced overfitting. The Transformer model, despite its architectural complexity, showed promising results by effectively capturing both short- and long-term dependencies through self-attention mechanisms, though it was more sensitive to hyperparameter configurations.

These findings suggest that deep learning models, especially GRU and Transformer, are well-suited for modeling the dynamic and nonlinear behavior of Bitcoin prices. Future research could explore hybrid architectures that integrate traditional time series components with attention-based deep learning mechanisms, the inclusion of exogenous variables such as macroeconomic indicators or social media sentiment, and the application of these models in real-time forecasting systems. Moreover, enhancing interpretability remains an important direction to support the practical deployment of these models in financial decision-making.

ACKNOWLEDGMENT

We want to start by expressing our sincere gratitude to ***Lecturer Duong Phi Long*** for always looking out for our team during this course. We value your support and sharing your invaluable experience with us because it will enable us to complete our project. Finally, we want to say how grateful that our team was able to attend your session and learn from you. Many thanks to ***Lecturer Duong Phi Long***. We sincerely appreciate your tireless efforts in leading our team to success and motivating us to keep moving forward. Your tireless efforts have been instrumental in ensuring that our project was a resounding success.

REFERENCES

[1] B. M. Henrique, V. A. Sobreiro, and H. Kimura, "Stock price prediction using ARIMA and ARIMAX models," *Expert Systems with Applications*, vol. 42, no. 6, p. 302–317, 2015, doi: 10.1016/j.eswa.2014.10.040.

[2] H. M, G. E.A., V. K. Menon, and S. K.P., "Predicting the price of Bitcoin using Machine Learning," *Procedia Comput. Sci.*, vol. 132, pp. 1351–1362, 2018, doi: 10.1016/j.procs.2018.05.050.

[3] R. McNally, J. Roche, and S. Caton, "NSE Stock Market Prediction Using Deep-Learning Models," *Proc.* 26th Euromicro Int'l *Conf.* Parallel, Distributed and Network-

Based Processing, pp. 339–343, 2018, doi: 10.1109/PDP2018.2018.00060.

[4] J. Nelson and J. P. Pereira, "GRU vs LSTM: A comparative study on cryptocurrency forecasting," 2020, arxiv: abs/2010.05755.

[5] A. Lim, Z. Zohren, and S. Roberts, "Time-series forecasting with deep learning: A survey," *Philosophical Transactions of the Royal Society A*, vol. 379, no. 2194, 2021, doi: 10.1098/rsta.2020.0209.

[6] Y. Kim and S. Kang, "Forecasting short-term Bitcoin price fluctuations using a hybrid ARIMA-LSTM model," *Applied Sciences*, vol. 12, no. 1, pp. 314, 2022, doi: 10.3390/app12010314.

[7] Sangarshanan, "Time series Forecasting — ARIMA models," *Medium*, Apr. 07, 2019. https://towardsdatascience.com/time-series-forecasting-arima-models-7f221e9eee06 (accessed Jun. 17, 2023).

[8] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," *Proc.* 22nd ACM SIGKDD Int'l *Conf.* Knowledge Discovery and Data Mining (KDD '16), *pp.* 785–794, 2016, doi: 10.1145/2939672.2939785

[9] F. M. Bianchi, E. Maiorino, M. C. Kampffmeyer, A. Rizzi, and R. Jenssen, *An overview and comparative analysis of Recurrent Neural Networks for Short Term Load Forecasting*. 2017. doi: 10.1007/978-3-319-70338-1.

[10] "Gated Recurrent Unit (GRU) With PyTorch," *FloydHub Blog*, Jul. 22, 2019. https://blog.floydhub.com/gru-with-pytorch/ (accessed Jun. 17, 2023).

[11] A. Vaswani et al., "Attention Is All You Need," arXiv, Jun. 12, 2017. https://arxiv.org/abs/1706.03762.