Stock Volatility Forecasting with Transformer Network

by

GOLNAZ SABABIPOUR ASL

A thesis submitted to
The Faculty of Graduate Studies of
The University of Manitoba
in partial fulfillment of the requirements
of the degree of

Master of Science

Department of Computer Science
The University of Manitoba
Winnipeg, Manitoba, Canada
December 2023

Thesis advisor

Ruppa Thulasiram

Author

GOLNAZ SABABIPOUR ASL

Stock Volatility Forecasting with Transformer Network

Abstract

Financial world faces many uncertainties due to decisions by governments, business entities, technology trends, natural calamities etc. Stocks and stock market form one of the major activities in financial world. Volatility is one of the main measures of uncertainty in financial stock markets. Hence, forecasting stock volatility is a critical component in many financial problems such as financial risk management, optimizing portfolios etc. In addition to traditional statistical techniques, there have been few artificial intelligence (AI) and machine learning (ML) techniques used in the literature for this volatility forecasting problem. Transformer Network (TN) architecture is one of newest ML techniques. For this thesis work, we utilized TN architecture with multi-head attention (MHA) mechanism for stock volatility forecasting. To enhance the performance of the TN, we incorporated different variations of the feed forward layer. The performance of three distinct TN models was evaluated by implementing three different deep learning (DL) layers (CNN, LSTM, and a hybrid layer CNN LSTM) in the encoder block of TN as the feed forward layer. The results clearly demonstrate that the TN model with the hybrid layer (CNN-LSTM) outperformed the other models, including a recently proposed data-driven approach. Furthermore, we assessed the performance of another latest model that is on built on TN known

as Informer model on a minute-scale Bitcoin dataset across various forecast lengths. Our findings underscore the advantages of the Informer model, specifically its ProbSparse attention mechanism and distilling operation, which substantially enhances its efficiency in handling long sequence time-series forecasting (LSTF) tasks.

Contents

List of Figures

List of Tables

Acknowledgments

I would like to begin by thanking my advisor, Dr. Ruppa Thulasiram, as well as the esteemed members of my committee, including Dr. Sara Rouhani and Dr. Aerambamoorthy Thavaneswaran, my partner, family and everyone who have supported me along the way.

This thesis is dedicated to my family.

Chapter 1

Introduction

The stock market holds significant importance in the economic development of modern society, which is an indicator of the financial health of a country [3]. Investing in the stock market gives investors a chance to profit while Investors' greatest concern is the volatility of the stock market [4]. Prediction of stock volatility is a crucial aspect in studying risk management, asset pricing, and several financial behaviors [5].

Predicting volatility is one of the biggest challenges for financial modelers. This is because stock data are highly nonlinear, extremely complicated, and exhibit a high degree of temporal variability. Traditional linear models like the autoregressive model (AR) and autoregressive integrated moving average (ARIMA) have some limitations like assumptions regarding the consistent variability of price data. One of the widely used models in volatility forecasting is generalized autoregressive conditional heteroscedasticity (GARCH) that was implemented by [6] by adding the variance lag term to the ARCH model [7]. Limitation of GARCH models is that it imposes

parameter restrictions, which could limit the dynamics of the conditional variance process. Different combinations of models based on GARCH have been proposed for volatility prediction and risk measurement.

The use of ML models has been popular in recent years for stock market prediction [8]. Because financial markets are very complicated, using DL techniques for forecasting is one of the most attractive methods [9]. Unlike statistical models, DL models offer a different approach. They learn directly from data through a process called training. By feeding these models with large amounts of data, they can uncover complex patterns and make predictions that might be too subtle for standard statistical methods to detect. This ability to learn from data makes DL models particularly powerful for forecasting the markets. For forecasting the market, the two most popular methods are recurrent neural network (RNN) [10], and long short-term memory (LSTM) [11]. Each of them has its own limitations. RNNs suffer from the problem of vanishing gradient. LSTM models were introduced as a solution to address the vanishing gradient issue in the RNN model. Liu [12] used LSTM network for volatility prediction and the result of this research demonstrated that LSTM has better performance than traditional ML models. While LSTM networks possess a significant memory capacity, more information tends to be forgotten when extending the transmission distance. This issue heightens the risk of failure [13]. To overcome those limitations, Vaswani et al. [1] introduced a novel DL architecture called transformer network (TN) that employs the attention mechanism and accomplishes outstanding results in solving natural language processing (NLP) issues [14].

In light of the outperformance of TN in modeling sequential data in NLP, it is worth

to apply them to time series data prediction as well. Some research has evaluated the performance of TN on stock market prediction, including price prediction, price movement, and volatility forecasting. One of the focus areas of existing research is the use of TN in sentiment analysis. Koksal and Ozgur [15] applied TN for predicting price movement by using textual information like financial news and comments from social media as input data.

Forecasting volatility has a crucial role in financial application [16]. Due to the importance of this topic, different methods have been compared to each other to get a reliable model with higher accuracy. DL models, especially RNN-based models for time series problems, have been shown to be a good method for forecasting volatility. RNN-based models have some limitations. TN were introduced to address the problems with previous RNN-based models. The aim of my research is to use a TN for volatility forecasting.

In recent years, the field of NLP has witnessed remarkable advancements, with the introduction of the TN architecture revolutionizing the way we approach various language-related tasks. The TN architecture, as introduced by models like bidirectional encoder representations from transformers (BERT) [17], has demonstrated exceptional capabilities in understanding and generating human language. However, like any technology, TN are not without their limitations. One notable challenge is their inefficiency in handling long texts, often leading to excessive computational costs [2]. To address this limitation, researchers have developed a novel model called Informer, introduced by Zhou et al. [2] that builds upon the TN's strengths while mitigating some of its weaknesses. This research explores how TN and informer net-

work could improve the efficiency and effectiveness of volatility forecasting.

In the following, Chapter 2 explores relevant studies providing a background on various model architectures, including data-driven exponentially weighted moving average (DDEWMA), LSTM, TN, and Informer network and provides a comprehensive literature review. Chapter 3 details the research framework employed, covering basic statistics, redesign and implementation of TN, and the implementation of LSTF for Bitcoin data. Following this, Chapter 4 focuses on experimentation and evaluation of the TN model, discussing research approaches and presenting results of univariate and multivariate volatility forecasting. Chapter 5 focuses on the Informer network model, highlighting its benefits for effective sequence forecasting. Finally, Chapter 6 concludes the thesis by summarizing findings from both TN and Informer models, outlining potential future work, and offering insights into the overall contribution of the research.

Chapter 2

Literature Review and Background

2.1 Literature Review

Using DL techniques for forecasting has become popular due to their high performance as compared to traditional models such as GARCH models. Xu et al. [3] adopted a new hierarchical graph neural network (HGNN) to analyse the market state from a hierarchical perspective for stock type forecasting. For the purpose of graph construction, they included both historical sequence patterns and stock relationships in their analysis. The result demonstrated the outperformance of HGNN over other the state-of-the-art solutions including attentive long short-term memory (ALSTM) [18], graph convolutional network (GCN) [19], and graph attention network (GAT) [20] models. Wang et al. [9] used TN for predicting the value of stock index the in future. While some researchers used TN in sentiment analysis using financial news and comments in social media, they used the daily closing price to predict stock market indexes. They evaluated the performance of TN model from two perspectives:

prediction accuracy and net value analysis. For prediction accuracy, they compared the results of mean squared error (MSE), mean absolute error (MAE), and mean absolute percentage error (MAPE). In comparison to traditional models, the TN had lower values for three indicators of prediction errors. They assessed how well the models performed in terms of net values by creating a basic trading strategy based on the predictions. They considered the passive strategy, buy & hold (B&H), as a benchmark. The results showed that in comparision with other models, TN model had the highest Sharpe ratio and total return. They performed the Mann–Whitney U test to display the extra profits of TN and the result demonstrated that the total return of TN was superior to other strategies. Jiang [21] provided a review of the works that used DL models for stock market prediction during 2017-2019 in order to provide a comprehensive understanding for researchers. In total, they covered 56 journal papers, 58 conference papers and 10 preprint papers. Papers were chosen that focused on the prediction of the closing prices of individual stocks and market indexes. They summarized all the articles and categorized and compared them from various perspectives. They provided several figures for readers that were easy to understand. Zhou et al. [2] discussed in their paper the need for accurate forecasting of long time series data, such as electricity consumption, and the challenges of using TN models for this purpose. To address these challenges, they introduced a novel model called Informer with three key features: (i) a ProbSparse self-attention mechanism for improved efficiency and performance, (ii) attention distillation to enhance the role of convolutional operators, and (iii) a generative style decoder for faster long-sequence predictions. Their experiments showed that Informer outperformed existing methods

in DL such as traditional RNN-based models, offering a solution for LSTF. Lu et al. [22] used Informer for stock market prediction and compared with LSTM, TN, and BERT on 1-minute and 5-minute data for various stocks and market indices, using standard MAE, root mean square error (RMSE), and MAPE as evaluation metrics. They concluded based on their results that Informer outperformed other networks on all datasets.

2.2 Background - Understanding Model Architectures

In the following, I briefly discuss the architecture of DDEWMA, LSTM and TN Network models.

2.2.1 Data-Driven Exponential Weighted Moving Average (DDEWMA)

A new and comprehensive data-driven EWMA (DDEWMA) forecasting model introduced in [23] is used to forecast the conditional volatility of log-returns directly. DDEWMA Volatility Forecasting Formula is:

$$\sigma_{t+1} = \frac{\alpha}{\rho}|R_t - \bar{R}| + (1-\alpha)\sigma_t \quad 0 < \alpha < 1 \qquad (2.1)$$

Here, α represents the best smoothing constant that we find by reducing the error error sum-of-squares in one-step-ahead forecasts, $E|R_t - \bar{R}| = \rho\sigma, \rho$ is the sign correlation of log-returns. The formula of sign-correlation is used to determine the

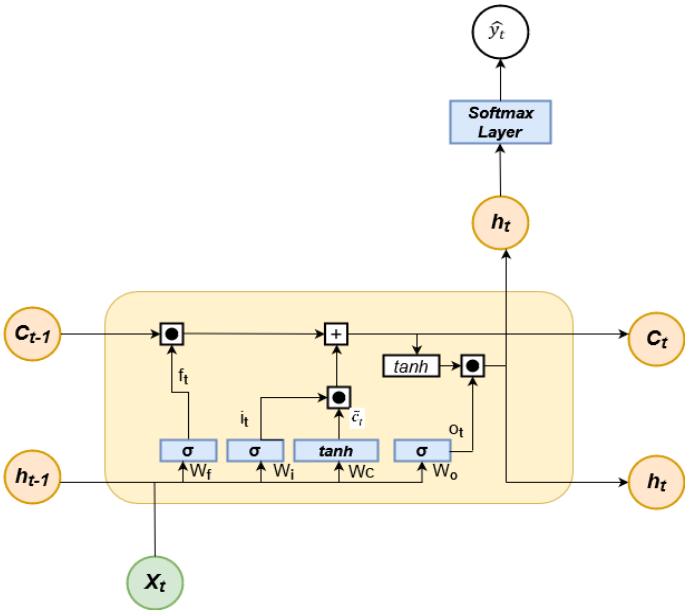

Figure 2.1: LSTM

distribution of observations with a random variable X and mean μ is:

$$\rho X = Corr(X - \mu, sign(X - \mu)) \qquad (2.2)$$

DDEWMA is an efficient model that can be used for any symmetric distribution.

2.2.2 Long Short-Term Memory (LSTM)

LSTM is a type of deep neural network that is part of the family of RNNs. [24]. LSTM introduced in [25] has a more complex architecture than a regular RNN. In response to the vanishing gradient issue that arises during standard RNN training, LSTM model was developed. It relies on memory cells and gates to retain information for long periods of time. Figure 2.1 presents the details of the LSTM architecture. LSTM's concept involves regulating the flow of information into and out of the network's memory cell using specific gate units: forget, input, and output. Actually, a

gate is a sigmoid (σ) [26] neural network layer followed by a pointwise multiplication operator. Each gate is controlled by the combination of the network's prior time step state h_{t-1} and the present input signal x_t. The forget gate determines which data will be eliminated from the cell state C_t, while the input gate determines what fresh information will be preserved within it. The output gate determines the updated state h_t. The subsequent equations describe the internal operations conducted within an LSTM neural unit.

$$f_t = \sigma(W_f[h_{t-1}, x_t] + b_f) \qquad (2.3)$$

$$i_t = \sigma(W_i[h_{t-1}, x_t] + b_i) \qquad (2.4)$$

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \qquad (2.5)$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \qquad (2.6)$$

where:

$$\tilde{C}_t = tanh(W_c[h_{t-1}, x_t] + b_c) \qquad (2.7)$$

$$h_t = o_t * tanh(C_t) \qquad (2.8)$$

In the above equation, x_t is the present input signal, f_t is forget gate, i_t is input gate, o_t is output gate, σ represents sigmoid function, C_t is the new cell state, $\tilde{C}_t$ is the updated cell state, h_t is the hidden state, W_f, W_i, W_o, and W_c represent weight matrices, while b_f, b_i, b_o, and b_c denote bias vectors, all of which are parameters that the network will learn during the training process [27].

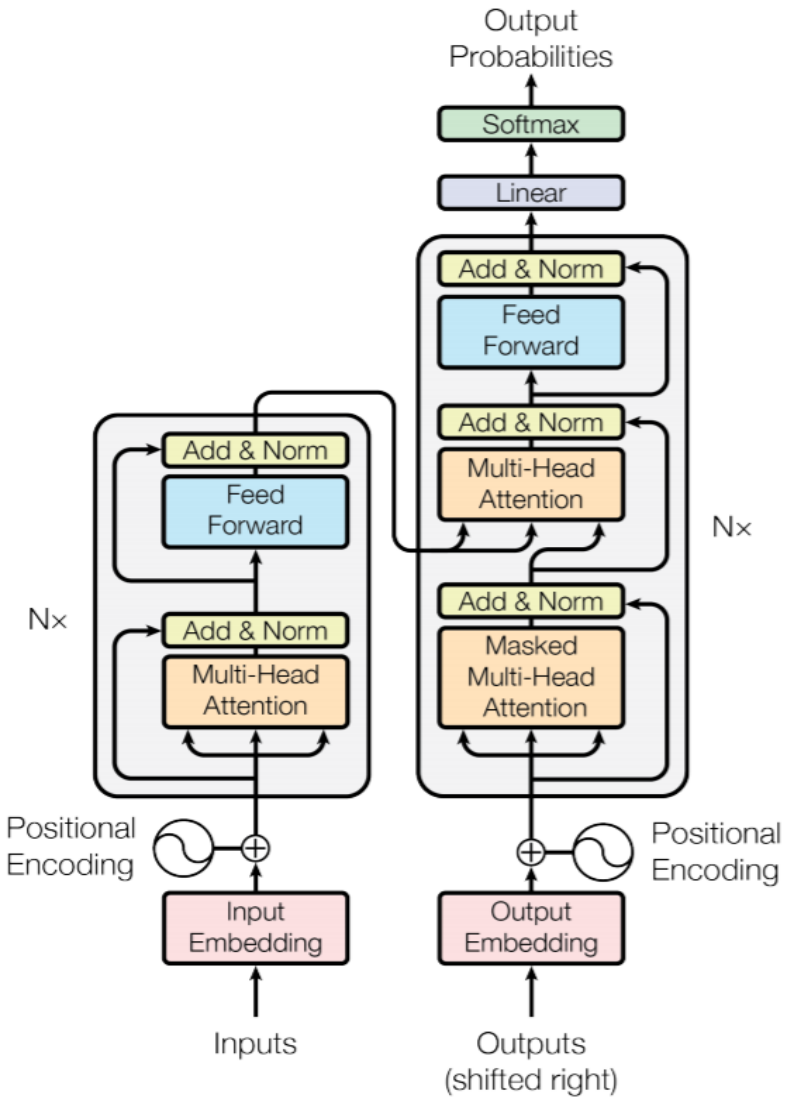

Figure 2.2: Architecture of TN, [1]

2.2.3 Transformer Network (TN)

This section provides an introduction to the TN architecture. Key components such as embeddings, positional encoding, encoder-decoder, multi-head attention, and feed-forward are discussed. Figure 2.2 presents the details of TN architecture.

Embeddings : The embedding layer transforms the input tokens and output tokens to vectors of dimension d_{model}. In the context of NLP, a token refers to a unit of text.

Positional Encoding (PE) : PE added to the embedded input in order to describe the sequential information within the time series [1]. The PE shares the same dimension, d_{model}, as the embeddings. This allows for both of them to be added together. Sine and cosine functions with different frequencies are employed to encode positional information:

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{model}}) \qquad (2.9)$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{model}}), \qquad (2.10)$$

where pos is the position and i is the dimension.

Encoder-decoder: As you can see in Figure 2.2, TN has the encoder-decoder architecture, that frequently applied in machine translation tasks [28]. The encoder-decoder architecture deals with long sequential data [29]. The encoder block consists of M layers, all having the same structure. Each layer comprises two sub-layers: a multi-head attention layer and a feed forward layer. Residual connections and normalization are employed in each sub-layer to enhance performance. The decoder block includes three sub-layers: a masked multi-head attention, a multi-head attention layer and a feed forward layer. In the decoder, the residual connection and normalization are used in each sub-layer to increase the performance.

Multi-head attention (MHA): In the MHA mechanism, every attention function runs in parallel with its corresponding projected versions of the query, key, and value ma-

trices. Subsequently, the outputs of all attention functions are merged together to generate the final result via a linear layer [1]. Concatenation is used to combine the outputs of multiple heads along the feature dimension. The formula for MHA is expressed as follows:

$$Multi - Head(Q, K, V) = Concat(head_1, head_2, ..., head_h)W^O \qquad (2.11)$$

where W^O stands for the weight matrix associated with the output, and $head_1$ to $head_h$ signify the outcomes of distinct attention heads as follows:

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \qquad (2.12)$$

where i = 1,2,..., h and QW_i^Q, KW_i^K, VW_i^V are weights of corresponding networks [9]. Feed Forward Network (FFN): Aside from the attention sub-layers, each layer within our encoder and decoder encompasses a fully connected feed forward layer that operates independently and uniformly on each position. This network consists of two linear transformations separated by a rectified linear units (ReLU) [30] activation.

$$FFN(x) = max(0, xW_1 + b_1)W_2 + b_2 \qquad (2.13)$$

While linear transformations are consistent at various positions, they employ varying parameters from one layer to another [1].

2.2.4 Informer: Transformer for Long Sequence Time-Series Forecasting

TN are seen as a big leap forward in the world of DL because they improve the reliability and precision of predictions. However, they face issues that prevent their

direct use in LSTF, being slow, using a lot of memory, and having a built-in limitation in their design. This led to the creation of a more efficient transformer-based model called Informer that was first introduced by [2]. A quick overview of Informer model is presented in Figure 2.3 . To comprehend the enhancements made by Informer over the basic TN [1], it is important to review the limitations of the original model:

- Quadratic computational complexity of original self-attention: The base TN exhibits a computational complexity of $O(T^2)$, where T represents the length of the time series. This complexity can be really hard to deal with when trying to solve the LSTF problem. Informer came up with a clever solution by using a new method called ProbSparse attention. This new method makes things much easier and quicker, especially when it comes to how much time and computer memory is needed. It offers a time and space complexity of $O(TlogT)$.

- Memory bottleneck during layer stacking: When we want to add N encoder/decoder to the basic TN, the memory usage becomes $O(NT^2)$, and this makes it hard to work with really long sequences. However, Informer employs a distilling operation that makes the memory needed much less. This way, it can handle long sequences more efficiently. As a result, it efficiently lowers total memory consumption to $O(N * TlogT)$.

- The speed plunge in predicting long outputs: Initially, the step-by-step prediction with the basic TN is as slow as using an RNN-based model. To address this issue, Informer employs a generative-style decoder. In this approach, instead of selecting a specific symbol as the start token, the Informer model chooses a 'shorter' segment from the input data that precedes the output sequence it

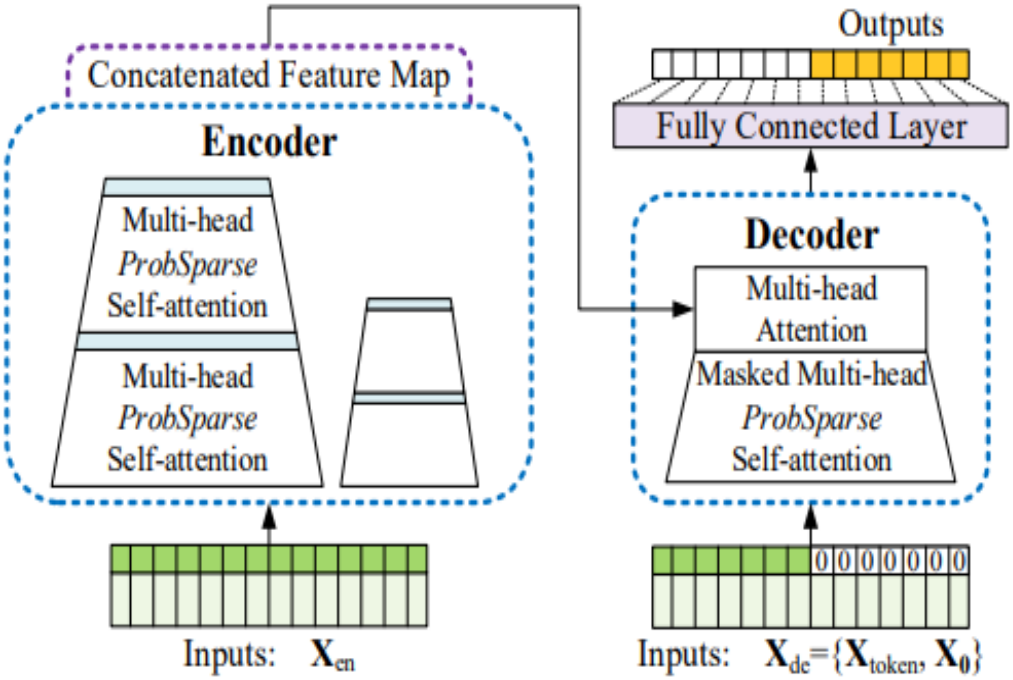

Figure 2.3: Architecture of Informer, [2]

wants to predict. For instance, if it aims to predict 480 data points (like what will happen in the next 5 days in a dataset), it begins by considering the 5 days of data it already has. This is referred to as the start token [2].

2.2.4.1 Input Representation

The input encoding technique of the Informer model is created to capture both the overall hierarchical time information (such as week, month, and year) and the independent time information (such as holidays and other events), which are essential for forecasting LSTF data. The original self-attention mechanism doesn't use this information properly, causing a mismatch between the queries and keys in the encoder

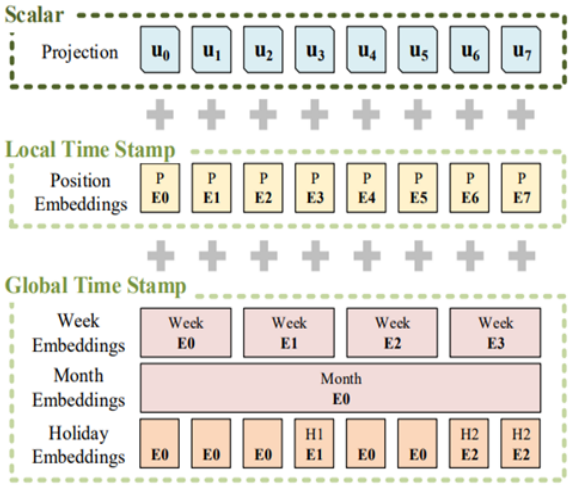

Figure 2.4: The input representation of Informer [2]

and decoder. These differences have an adverse effect on the accuracy of the predictions. Figure 2.4 shows the input representation of Informer. The input embedding is made up of three different pieces, a local time stamp (Position), scalar projection and global time stamp embeddings (Minutes, Hours, Week, Month, Holiday etc.).

For each value in the input sequence at time t, denoted as x_i^t, the encoding method first transforms it into a d_{model}-dimensional vector u_i^t through 1-D convolutional filters. Here, d_{model} represents the dimensionality of the input encoding.

Next, a position embedding is applied at time t to preserve the local context, which is defined as follows:

$$PE_{(pos,2j)} = sin(pos/(2L_x)^{2j/d_{model}}) \qquad (2.14)$$

$$PE_{(pos,2j+1)} = cos(pos/(2L_x)^{2j/d_{model}}) \qquad (2.15)$$

where j $\in \{1, ..., [d_{model}/2]\}$ and $pos = 1, 2, \ldots, L_x$, L_x is the length of the input sequence. Every global timestamp (such as the hour, weekday, day, and month) is

utilized in conjunction with a learnable stamp embedding denoted as SE(pos) with a restricted vocabulary size (up to 60, namely taking minutes data as the finest granularity). This approach considers minutes as the finest granularity for timestamps. Thus, we have the feeding vector:

$$X_{feed[i]}^{t} = \alpha u_i^t + PE_{L_x \times (t-1)+i} + \Sigma_p [SE_{L_x \times (t-1)+i}] p \qquad (2.16)$$

where t-th sequence input x_i^t, $i \in \{1, \ldots, L_x\}$, p is types of global time stamps, and α is the factor helps balance the size between the scalar projection and the local/global embeddings. If the input sequence has been normalized, we recommend setting α to 1.

2.2.4.2 ProbSparse Attention

ProbSparse Attention is a novel self-attention mechanism introduced in the Informer model. It capitalizes on the observation that the scores generated by the traditional self-attention mechanism exhibit a long-tail distribution. In this distribution, there are active queries that significantly contribute to the attention computation, and lazy queries that generate trivial attention (these queries contribute very little or negligible information to the attention mechanism). To leverage this insight, ProbSparse Attention identifies the active queries and constructs a reduced query matrix, denoted as $Q_{reduced}$. This reduced matrix is then used to calculate the attention weights with a computational complexity of $O(TlogT)$, where T represents the length of the time series. This approach improves efficiency compared to the quadratic complexity $O(T^2)$ of the original self-attention mechanism (introduced by [1]). By effectively selecting and processing the active queries, ProbSparse attention

enhances the overall computational efficiency of basic TN, making it more suitable for LSTF tasks. The self attention formula in the original paper is:

$$Attention(Q, K, V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \qquad (2.17)$$

Therefore, the QK^T multiplication takes $O(T^2D)$ computational complexity. In ProbSparse attention, our goal is to create a new Q_{reduce} matrix and define:

$$Attention(Q, K, V) = \text{Softmax}\left(\frac{Q_{reduce}K^T}{\sqrt{d_k}}\right)V \qquad (2.18)$$

where Q_{reduce} is a sparse matrix of the same size of q and it exclusively contains the top queries based on the sparsity measurement $M(q, K)$.

2.2.4.3 Query Sparsity Measurement

Query Sparsity Evaluation (QSE) measures the sparsity of queries in a given matrix $M(q_i, K)$ to identify the active queries q_i for constructing the reduced query matrix Q_{reduce}. The prevailing $<q_i, K_i>$ pairs drive the probability distribution of the active queries q_i away from the uniform distribution. The likeness between distribution can be used to distinguish the important queries. The i-th query's sparsity measurement is defined as [2]:

$$M(q_i, K) = \ln\left(\sum_{j=1}^{L_K} e^{\frac{q_i k_j^T}{\sqrt{d}}}\right) - \frac{1}{L_K}\sum_{j=1}^{L_K}\frac{q_i k_j^T}{\sqrt{d}} \qquad (2.19)$$

When the i-th query yields a higher value for $M(q_i, K)$, it leads to a higher attention probability p and is more likely to include the dominant dot-product pairs within the header field of the long-tail self-attention distribution. The important thing to understand here is when $M(q_i, K)$ is larger, the query q_i should be in Q_{reduce} and vice

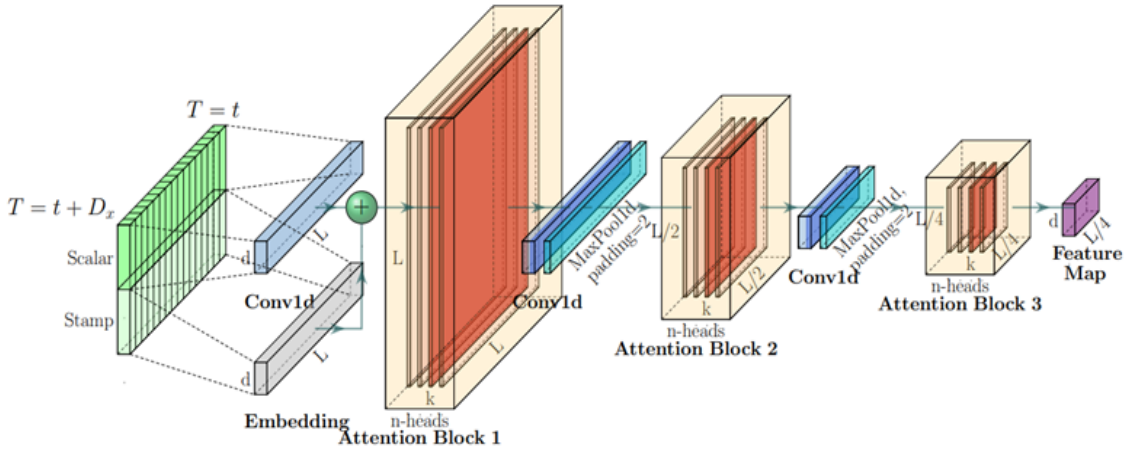

Figure 2.5: The single stack in Informer's encoder [2]

versa. Zhou et al. [2] used KL-based measurement (Kullback-Leibler divergence), and is one specific formulation derived from entropy that is commonly used to compare two probability distributions.

2.2.4.4 Self-attention Distilling

The distilling operation is used to reduce the input size between encoder layers into its half slice. Due to the inherent nature of the ProbSparse self-attention mechanism, the encoder's feature map contains redundant combinations of the value V. We use the distilling operation as depicted in Figure 2.5to emphasize the important ones with predominant features and make a focused self-attention feature map in the following layer. Distilling procedure moves from j-th layer into $(j+1)$-th layer [2]:

$$X_{j+1}^{t} = \text{MaxPool}(\text{ELU}(\text{Conv1D}([X_j^t]_{AB}))) \qquad (2.20)$$

where $[.]_{AB}$ illustrates the attention block. It includes the multi-head ProbSparse self-attention and the fundamental operations. Here, $Conv1d()$ carries out a 1-D

convolution using filters with a kernel width of 3 on the time dimension, utilizing the Exponential Linear Unit (ELU) activation function[31]. We incorporate a max-pooling layer with stride 2 and downsample X_t into its half slice after stacking a layer, which minimizes memory consumption to be $O((2-\epsilon)TlogT)$, where ϵ represents a small value. To make the distillation operation more robust, we create copies of the main stack with smaller inputs and gradually reduce the number of self-attention layers at a time. They combine the results from all the stacks to get the ultimate hidden representation of the encoder.

One of the significant issues in finance is volatility forecasting and it plays an important role in financial application [16]. Due to importance of this topic, different methods have been introduced and compared to each other to get a reliable model with higher accuracy. Neural networks (NN) models, especially RNN-based models for time series problems have been shown to be a good method for forecasting volatility. RNN-based models have some limitations. TN were introduced to address the problems with previous RNN-based models. The aim of this research is to use TN for stock volatility forecasting.

Research contributions of this work to the field include:

1. *Feature importance analysis for volatility forecasting*: identifying the most crucial feature or combination of features for forecasting stock volatility is achieved through feature importance analysis

2. *Introduction of TN with variations of feed forward Layer*: The research proposes a TN and enhances the model's performance by incorporating three variations of the feed forward layer: convolutional neural network (CNN), LSTM, and a

hybrid approach combining both (CNN-LSTM). This is a novel approach in combining different NN architectures for improved forecasting.

3. *Effectiveness of hybrid layer (CNN-LSTM) as feed forward Layer*: Experimental results indicate that the hybrid CNN-LSTM layer as the feed forward layer in the TN outperforms other models. This suggests the effectiveness of combining CNN and LSTM for financial volatility forecasting within the TN.

4. *Implementation of univariate and multivariate volatility forecasting*: With the understanding of importance of feature selection for improved forecasting accuracy, we achieved further improvement by fine tuning and comparing univariate and multivariate features of the data

5. *Benefits and robustness of the Informer model in LSTF*: The study observes that the Informer model maintains stable forecasting performance, especially when the forecast horizon is short. The model exhibits moderate growth in execution time with increased forecast length, indicating its robustness in addressing computational complexity and memory bottlenecks.

In the next chapter we present our research framework that covers basic statistics, implementation of TN, and the implementation of LSTF for Bitcoin data.

Chapter 3

Transformer Network Algorithms

In this chapter we describe the modifications to the TN for our problem.

3.1 Basic Statistics and Time Embedding in TN

The historical price of S&P 500 index was utilized to measure the efficiency of the proposed methodology. The dataset used in this study consisted of daily data of the S&P 500 index from 1982 to 2023, downloaded from Yahoo! Finance. It comprised of 10,337 rows and 7 columns, namely Date, High, Low, Open, Close, Adj Close, and Volume. The historical data Adj Close price spanning from 1982 to 2023 presented in Figure 3.1 depicts a comprehensive overview of the price trends and movements in the S&P 500 index.

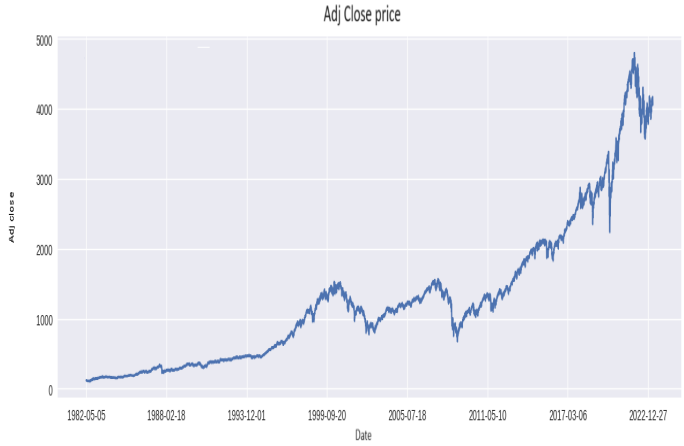

Figure 3.1: historical data S&P 500 between 1982-2023

Figure 3.2: return of the S&P 500 between 1982-2023

3.1.1 Basic Statistics

The Adj Close price is used to calculate return volatility. The Adj Close price informs investors of required reaction after a corporate action. Figure 3.2 shows return of S&P 500. The dataset for analysis divided with 70% allocated for training, and 15% each for validation and testing, as illustrated in Table 3.1. For generating

Table 3.1: Dataset Description

Data	Training data	Validation data	Test data	Sliding Window
Daily data of S&P 500	7233	1550	1549	12

Table 3.2: Summary of the dataset

	High	Low	Open	Close	Adj Close	Volume
count	10337	10337	10337	10337	10337	10337
mean	1290.9	1275.04	1283.2	1283.5	1283.5	1.999655e+09
std	1061.1	1048.6	1055.1	1055.2	1055.2	1.897309e+09
min	103.01	102.1	102.4	102.4	102.4	1.499000e+07
25%	417.2	413.7	416.04	416.04	416.04	2.229700e+08
50%	1127.5	1114.8	1121.3	1121.3	1121.3	1.335400e+09
75%	1544.02	1529.1	1536.9	1538.5	1538.5	3.580900e+09
max	4818.6	4780.04	4804.5	4796.5	4796.5	1.145623e+10

input sequences, we employed a sliding window approach with a window size of 12 instances. Each input sequence paired with a corresponding output forecasting, which consisted of a sequence of length 1. To ensure effective model convergence, the data were normalized using min-max scaler [32], which scales the values to a range of [0,1]. Normalizing the data facilitates faster convergence of the model and ensures that all features are within a consistent range for training and inference purposes. Table 3.2 presents a comprehensive description of the dataset used in this study.

The dataset used in this study showed that mean and the standard deviations for the Open, Close, and Adj Close columns are very similar, indicating that the variability or dispersion of these values is comparable. However, the results derived from employing the random forest algorithm on the time series data demonstrated that the Open price and Adj Close price were two most significant factors in the time series data. The feature importance scores indicate the relative importance of each

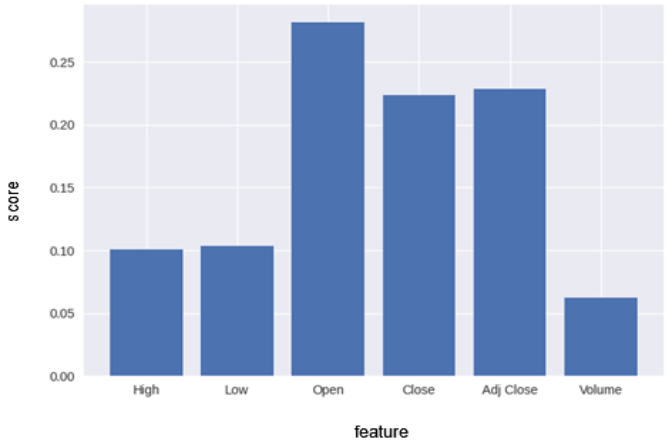

Figure 3.3: Feature importance

attribute in predicting outcomes. Notably, the Open attribute received the highest score, indicating its predominant influence on the predictive model. On the other hand, the Volume attribute received the lowest score, suggesting a weaker effect on the model's predictive capabilities. It is important to note that while the feature importance scores for all attributes were below the score of 0.5, the Open attribute stood out with a score of 0.28, indicating its relatively higher influence compared to other attributes.

Two key features identified as the most important in the feature importance plot. To compute volatility, first the Open price was considered, second the Adj Close price, and third the mean of these two values (Open and Adj Close prices). By measuring volatility, valuable insights into the fluctuations and potential risks associated with the S&P 500 index were gained.

The volatility of a stock is seen as an indicator of uncertainty. Volatility is a statistical measure that shows how much the returns of a particular investment or market index

vary. Generally, higher volatility indicates greater risk associated with the financial asset. In this research, we propose to use the TN for volatility forecasting instead of stock price prediction. The volatility σ of a stock is a measure of the uncertainty regarding the returns provided by the stock. Several models (and formulas) play crucial roles in computing volatility, each with its unique advantages and accuracy. In our research, we employed the fundamental formula based on standard deviation to compute volatility. The formula of standard deviation $\sigma = \sqrt{\frac{\sum(X_i-\mu)^2}{N}}$ where N is size of population, μ is the mean of population, and X_i is each value from population with a window size of 5. This involved analyzing a sliding window of five consecutive values of the Open price and applying three distinct TN models to forecast volatility. In the next step, Adj Close price was used to compute volatility and implemented three different TN models. Lastly, volatility was computed using the mean of the Open and Adj Close prices, again employing three different TN models. Figure 3.4 captures the volatility computed based on Adj Close price of the S&P 500, providing a clear representation of the fluctuations in the S&P 500 index.

The opening and closing prices of a stock reflect the first and last transactions during a trading day, respectively, but they can be influenced by various factors and may not fully capture the stock's true value. However, the Adj Close price offers a more accurate representation for stock forecasting. In addition to supply and demand fluctuations, Adj Close price metric incorporates other important factors such as dividends, stock splits, and corporate actions that can impact a stock's price. By adjusting for these factors, the Adj Close price provides a comprehensive view of the stock's value at the end of the trading day. Utilizing the Adj Close price

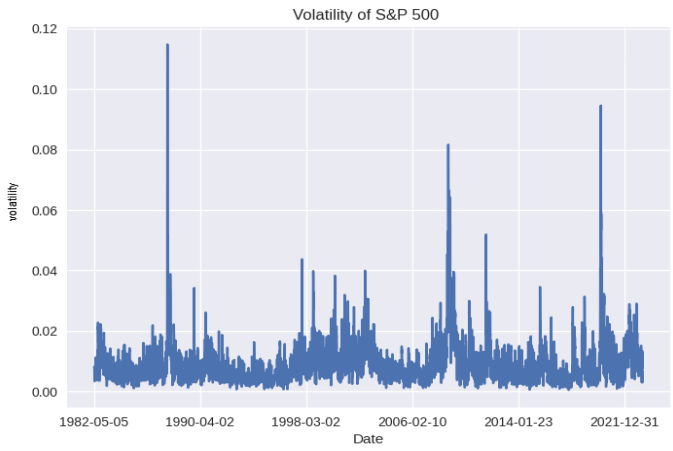

Figure 3.4: Historical Volatility of S&P 500 between 1982-2023

in stock forecasting allows to consider the entirety of the trading day's activity and incorporate all relevant market information, leading to a more accurate assessment of a stock's performance. In this research, the effectiveness of different price has been assessed in forecasting stock volatility by utilizing TN model.

3.1.2 Time Embedding

To establish the temporal relationships within each input sequence, the TN requires embedded time vectors. Several studies have utilized Time2Vec to produce time-embedded vectors for forecasting models in time series forecasting [33], [34]. Time2Vec, introduced in [35], plays a crucial role in implementing the TN for time-series data. It offers a vector representation of time that encompasses both periodic and non-periodic patterns, ensuring invariance to time rescaling. The findings of the study [35] demonstrated that replacing the concept of time with its Time2Vec representation improves the performance of the final model across various models and

problems. Time2Vec denoted as $t2v(\tau)$, is mathematically computed as follows [35]:

$$t2v(\tau) = \begin{cases} \omega_i \tau + \varphi_i & if \quad i = 0 \\ F(\omega_i \tau + \varphi_i) & if \quad 1 \leq i \leq K \end{cases} \tag{3.1}$$

The periodic feature consists of a linear function that is contained within a function F while the non-periodic feature is a linear function with slope ω and intercept φ. By passing the input data sequence through the $t2v(\tau)$ function both periodic and non-periodic features are computed. The features are merged with the input value sequence and fed to the TN model.

3.1.3 Transformer Network

To enhance the model's flexibility, in this study the PE is replaced with Time2Vec. The TN model offers a unique methodology by avoiding recursion and instead relies extensively on the attention mechanism to capture overall connections between input and output. Embeddings layer, PE, encoder-decoder, MHA, and feed forward layer are important components of TN. To enhance the model's flexibility, we made some modifications to the original TN architecture. First, in this study, we substituted PE with Time2Vec. Second, we modified the decoder to better align with our specific requirements. In the original TN design, both the encoder and decoder blocks consist of layers: a MHA layer, a normalization layer, and a feed forward layer. In this research, the decoder section has significant alterations, incorporating global average pooling, two dropout layers, and two dense layers, with the final dense layer yielding the output. The encoder unit comprised of MHA, normalization layers, and feed forward layers. The novelty of this study is that three different variations of the feed

forward layer was used, (namely CNN, LSTM, and a hybrid approach combining both (CNN-LSTM)) instead of a simple linear layer. When LSTM was used as the feed forward layer, the first LSTM layer consisted of 256 units, both LSTM layers utilized the hidden layer activation function (tanh) [36]. The input to the encoder involved embedding the input sequence using Time2Vec vectorization. The encoder produced a fixed-length output, before feeding into the decoder. Ultimately, the final dense layer in the decoder is responsible for generating the model's output.

Figure 3.5 demonstrated the TN Model of this research by applying LSTM layer as feed forward layer. There are different hyperparameters in TN. The mean squared error was used as the loss function. The Adam optimizer [37] was used to train the model. In order to prevent overfitting, a dropout technique with a rate of 0.1 was utilized for each sub-layer. Models were evaluated with MSE, MAE, and MAPE metrics. Batch-size (the number of training examples utilized in one iteration) in this research was 32, dimension of key and value for attention layers were 256, and 8 heads were employed for MHA layer. The encoder block was made up of 3 stack layers. All models were fit with 50 epochs.

3.2 Transformer Network Algorithms

A clear and concise representation of the algorithm designed for this research problem is presented in Algorithms 1 and 2. S_L refers to the length of the input sequence ($S_L = 12$) and d_m is dimension of the model. MHA involves the use of multiple attention functions running in parallel. In MHA, there are h attention layers that run simultaneously (h is equal to 8). Algorithm 1 demonstrates the MHA

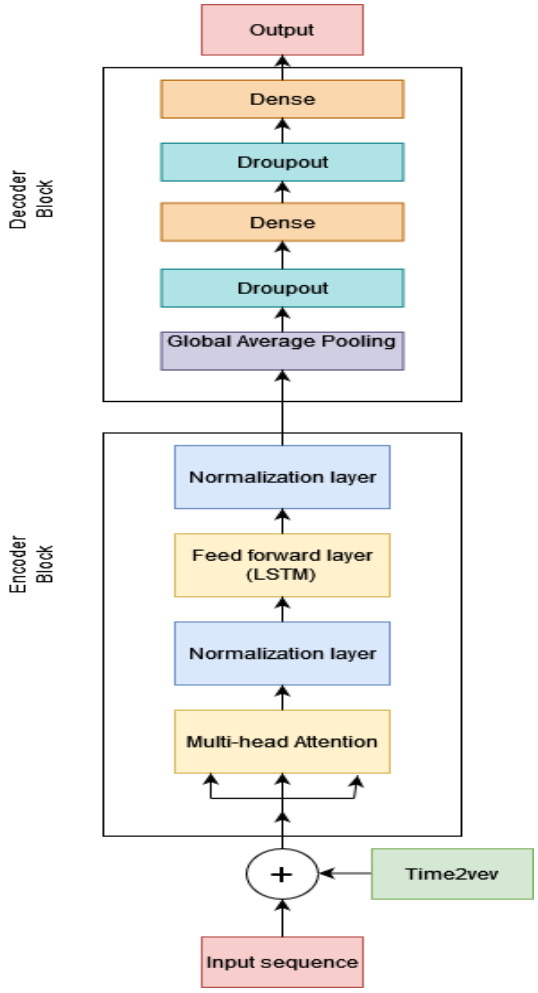

Figure 3.5: Implementation of TN Model by applying LSTM layer as feed forward layer

function.

Processing financial data using an encoder and decoder block for forecasting stock volatility is outlined in Algorithm 2. The first step is to compute the return of a stock. After that the volatility of the stock is computed. Dataset is split into training, validation, and testing data. After applying Tme2Vec method, the encoder block processes the input data applying a MHA mechanism to the data, then passing the result through a feed forward layer for which three different variations of the feed

Algorithm 1 Multi-head attention (MHA) Function

Multi-head Attention Layer (q,k,v)

Input: tensor q,k,v $\in R^{d_m \times S_L}$

output: tensor $y' \in R^{d_m \times S_L}$

$$q', k', v' = W_q x + b_q, W_k x + b_k, W_v x + b_v$$

$$q' = \begin{bmatrix} \mathrm{q}'_1 \\ \vdots \\ \mathrm{q}'_h \end{bmatrix}, \quad k' = \begin{bmatrix} \mathrm{k}'_1 \\ \vdots \\ \mathrm{k}'_h \end{bmatrix}, \quad v' = \begin{bmatrix} \mathrm{v}'_1 \\ \vdots \\ \mathrm{v}'_h \end{bmatrix}$$

$$y = \begin{bmatrix} Attention(\mathrm{q}_1, k_1, v_1) \\ \vdots \\ Attention(\mathrm{q}_h, k_h, v_h) \end{bmatrix}$$

$$y' = W_y y + b_y$$

return y'

forward layer, (namely CNN, LSTM, and a hybrid approach combining both (CNN-LSTM)) instead of relying solely on a simple linear layer have been incorporated. This is repeated for 3 times using the EncoderUnit function. The decoder block takes the output of the previous encoder block with global average pooling, two dropout layers, and two dense layers, with the final dense layer yielding the output.

Evaluation criteria used in this research in order to compare the result of TN are MSE, MAE, and MAPE. MAE is an average of the absolute difference between the actual and predicted values. It reflects the absolute errors in forecasting [38].

$$MAE = \frac{1}{N} \sum_{i=1}^{N} |\hat{y}_i - y_i| \qquad (3.2)$$

where $\hat{y}_i$ is the predicted value, y_i is the true value, and N is the sample size. MSE is

Algorithm 2 Transformer Network is used for Stock Volatility Forecasting

Data: Adj Close price of stock, $P_t, t = 0, ..., k, ...T_1$

1. Compute return, volatility

$r_t \leftarrow \frac{P_t - P_{t-1}}{P_{t-1}}$, $t = 1, ..., T_1$

$vol_t \leftarrow$ Standard Deviation of r_t

2. Normalize data, split data

Use Min-max normalization to normalize vol_t

Split the data vol_t sequentially into 70% train, 15% validation, 15% test

3. Encoder input data

training data are separated into sliding window sequences with a length of 12 days

(seq-len = 12)

for i in range(seq-len, len(train-data)) **do**

X-train.append(train-data [i - seq-len :i])

Y-train.append(train-data[i])

end for

Apply the Time2Vec method to generate PE for each X[t], i.e., t2v(X[t]) $\in R^{d_m \times S_L}$.

4. Encoderblock

Input: tensor x $\in R^{d_m \times S_L}$

output: tensor x $\in R^{d_m \times S_L}$

Function EncoderUnit(x):

$x' =$ layerNorm(x + Multi-head Attention (x, x, x))

$z' =$ layerNorm(x' + feedforward(x'))

return z'

for $i = 1, \ldots N$ **do**

x = EncoderUnit(x)

end for

return x

5. Decoderblock

Input: tensor x $\in R^{d_m \times S_L}$

output: tensor $z' \in R^{d_m \times 1}$

Function DecoderUnit(x):

x' = Global Average Pooling(x)

x' = Dropout(x')

x' = Dense(x')

x' = Dropout(x')

z' = Dense(x')

return z'

an average of the squared difference between the actual and predicted values.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2 \qquad (3.3)$$

where $\hat{y}_i$ is the predicted value, y_i is the true value, and N is the sample size. MAPE is relative indicator to measure the percentage of errors between the actual and predicted values [39].

$$MAPE = \frac{1}{N}\sum_{i=1}^{N}|\frac{\hat{y}_i - y_i}{y_i}| \times \%100 \qquad (3.4)$$

where y_i represents the actual value of volatility, $\hat{y}_i$ stands for the estimated value for volatility and N represents the number of samples in the dataset.

These three indicators assess how accurately models make predictions from various viewpoints. The smaller values of three of them mean better performance.

3.3 Implementing Long Sequence Time-Series Forecasting for Bitcoin Data

The minute-scale Bitcoin data was used to implement LSTF model. Minute-scale data offers a more intricate and timely perspective of market movements, a necessity for various trading strategies, risk management, and real-time decision-making within financial markets. The data downloaded from Kaggle, the Bitcoin data for the year 2021 at a one-minute scale. The dataset covers the period from January 1, 2021, at 12:06 AM to January 13, 2021, at 9:50 AM, including a total of 17,750 one-minute data points. These data points were divided into training, validation, and testing datasets as follows:

- Training: 12,435 data points
- Validation: 1,765 data points
- Test: 3,550 data points

The dataset consists of 7 columns: date, low, high, open, close, volume BTC, and volume USD. The columns in this dataset include:

- Date: This timestamp is in UTC timezone.
- Open: The opening price of the given time period.
- High: The highest price during the time period.
- Low: The lowest price during the time period.
- Close: The closing price at the end of the time period.
- Volume BTC: The volume transacted in the respective cryptocurrency. For instance, for BTC/USDT, this volume is in BTC.

- Volume USD: The volume transacted in the base/converted currency. In the case of BTC/USDT, this is in USDT, representing the total value of traded cryptocurrency converted to USDT.

In the next chapter we present our research framework that covers experiments and evaluation of TN model.

Chapter 4

Experiments and Evaluation of Transformer Network Model

In this Chapter we focus on experimentation and evaluation of the TN model, discussion on our research approaches and presenting results of univariate and multivariate volatility forecasting.

4.1 Dataset and Research Approaches

The first approach in volatility forecasting involves using only the volatility feature as shown in Table 4.1. There are three variations within this approach: (i) using the Open feature to compute volatility, (ii) using the Adj Close feature to compute volatility, and (iii) using the mean of the Open and Adj Close features to compute volatility.

The second approach in volatility forecasting involves utilizing all available features.

Table 4.1: Overview of the datasets

	High	Low	Open	Close	Adj Close	Volume	Volatility
mean	1290.9	1275.04	1283.2	1283.5	1283.5	1.999655e+09	0.0092
std	1061.1	1048.6	1055.1	1055.2	1055.2	1.897309e+09	0.0070
min	103.01	102.1	102.4	102.4	102.4	1.499000e+07	0.0005
25%	417.2	413.7	416.04	416.04	416.04	2.229700e+08	0.0051
50%	1127.5	1114.8	1121.3	1121.3	1121.3	1.335400e+09	0.0075
75%	1544.02	1529.1	1536.9	1538.5	1538.5	3.580900e+09	0.0112
max	4818.6	4780.04	4804.5	4796.5	4796.5	1.145623e+10	0.1147

In this method, for computing volatility, two approaches are considered: (i) using the Adj Close feature and (ii) using the Open feature. These approaches differ in terms of the number of features used for volatility forecasting and the data employed for computing volatility.

4.2 Univariate Volatility Forecasting

4.2.1 Using Open Feature for Computing Volatility

As previously stated, Open feature emerged as the most significant feature according to the feature importance analysis. All three models were compiled using MSE loss function, the Adam optimizer, and evaluated with MSE, MAE, and MAPE metrics. All models were fit with 50 epochs. Table 4.2 provides the corresponding outcomes that obtained the result of test data by employing three distinct layer configurations (CNN, LSTM, and a hybrid approach that combines both (CNN-LSTM)). In Figure 4.1, comparison of the actual volatility with the corresponding forecasted values is presented. This plot aims to provide a comprehensive assessment of the model performance in capturing the true fluctuations in volatility over time. Further-

Table 4.2: Comparative analysis of three models: univariate volatility forecasting

Models	MSE	MAE	MAPE
Transformer Network(LSTM layer)	1.1350e-05	0.0019	30.3417
Transformer Network(CNN layer)	4.6428e-05	0.0049	82.4246
Transformer Network(CNN-LSTM layer)	1.0810e-05	0.0018	29.5690

Figure 4.1: volatility forecasting

more, Figure 4.2 illustrates a plot of the MSE loss over 50 epochs. The smooth and progressively decreasing MSE loss plot indicates a well-optimized training process, where the model effectively adjusts its parameters to minimize forecasting errors. By examining the MSE loss plot in conjunction with the comparison of actual and forecasted volatility values, a more comprehensive evaluation of the model's performance is achieved.

4.2.2 Using Adj Close Feature for Computing Volatility

Alternatively, we employed the Adj Close feature instead of the Open feature for volatility forecasting with the same model configuration and evaluation metrics. The

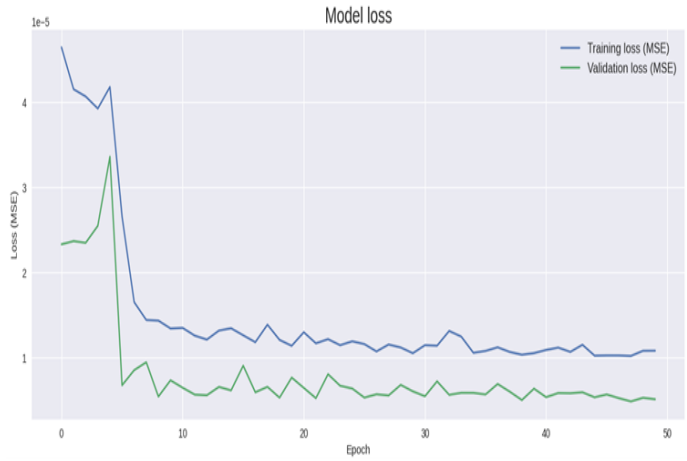

Figure 4.2: Model loss by applying TN (CNN-LSTM as feed forward layer)

Table 4.3: Comparative analysis of three models: univariate volatility forecasting

Models	MSE	MAE	MAPE
Transformer Network(LSTM layer)	1.1642e-05	0.0022	28.2502
Transformer Network(CNN layer)	0.0001	0.0051	68.5471
Transformer Network(CNN-LSTM layer)	1.0801e-05	0.0021	26.9638

resulting outcome is presented in Table 4.3, showing the performance of the three layers configurations: CNN, LSTM, and the hybrid CNN-LSTM approach. Figure 4.3 presents a comparison of the actual volatility value with the corresponding forecasted values. This plot aims to provide a comprehensive assessment of the model's performance in capturing the true fluctuations in volatility over time.

Additionally, a plot of MSE loss by applying 50 epochs is presented in Figure 4.4. A smooth and steadily decreasing MSE loss plot indicates a well-behaved training process, where the model is successfully optimizing its parameters to minimize the forecasting errors.

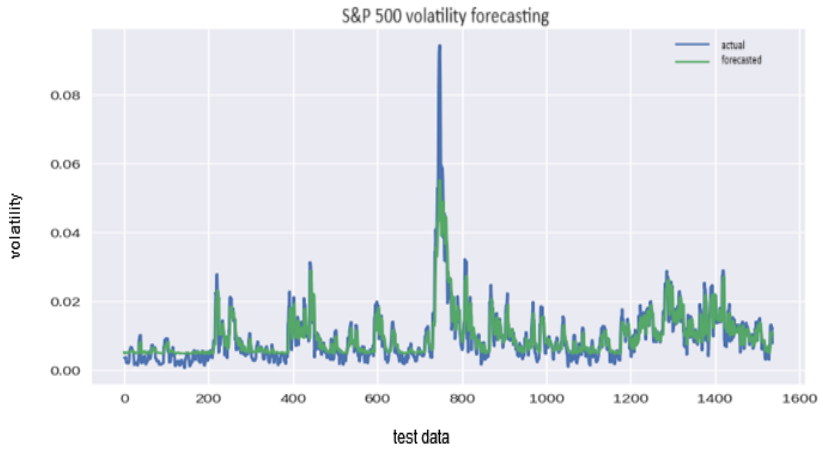

Figure 4.3: volatility forecasting

Figure 4.4: Model loss by applying TN (CNN-LSTM as feed forward layer))

4.2.3 Using the Mean of Open and Adj Close Features for Computing Volatility

The mean of the Open and Adj Close features were computed and used to compute the volatility. The same model configuration and evaluation metrics as before were utilized. All models are fit with 50 epochs. The resulting outcomes is presented in

Table 4.4: Comparative analysis of three models: univariate volatility forecasting

Models	MSE	MAE	MAPE
Transformer Network(LSTM layer)	2.0847e-05	0.0037	70.1799
Transformer Network(CNN layer)	2.1004e-05	0.0042	88.2180
Transformer Network(CNN-LSTM layer)	2.1006e-05	0.0037	75.7455

Table 4.4, displaying the performance of the three different TN models.

4.3 Multivariate Volatility Forecasting

4.3.1 Using Adj Close Feature for Computing Volatility

In this step, all seven features were used for forecasting volatility by implementing TN. To accommodate these differing input setups, modifications to the input sequence were made and adjusted the input shape within TN model. To enhance the model's performance, a moving average [40] technique employed on the dataset to reduce noise and minimize short-term fluctuations that are inherent in the time series data. The performance of three different models by applying different feed forward layer (CNN, LSTM, and hybrid (CNN-LSTM)) were assessed instead of the simple one. After compiling the model with the MSE loss, the Adam optimizer, and utilizing MSE, MAE, and MAPE as evaluation metrics, the models were fitted for 50 epochs. Table 4.5 presents results for the test data by employing three different feed forward layer configurations, (namely CNN, LSTM, and a hybrid approach combining both (CNN-LSTM)). Figure 4.5 demonstrates the comparison of the actual volatility and corresponding forecasted values, utilizing all columns as input for the model. Fig-

Table 4.5: Comparative analysis of three models: multivariate volatility forecasting

Models	MSE	MAE	MAPE
Transformer Network(LSTM layer)	4.7215e-06	0.0018	35.3797
Transformer Network(CNN layer)	2.1032e-05	0.0032	75.5001
Transformer Network(CNN-LSTM layer)	4.0552e-06	0.0016	33.0305

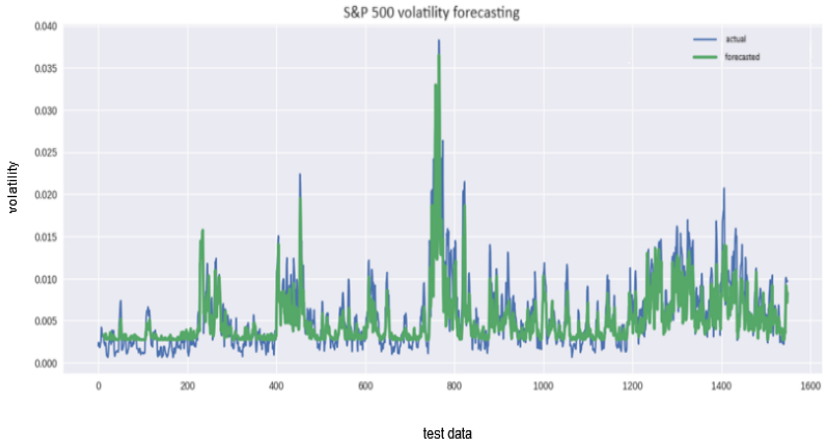

Figure 4.5: volatility forecasting

ure 4.6 illustrates the MSE loss plot obtained from training the TN model with all seven features over 50 epochs. A smooth and consistent decrease in the MSE loss signifies a successful training process, indicating that the model effectively adjusts its parameters to minimize forecasting errors. By examining the MSE loss plot in conjunction with a comparison of the actual and forecasted volatility values, a more comprehensive evaluation of the model's performance is achieved.

4.3.2 Using Open Feature for Computing Volatility

In this step, instead of using the Adj Close feature to compute volatility, Open feature was used. The rest of the steps, including incorporating seven features for

Figure 4.6: Model loss by applying TN (CNN-LSTM as feed forward layer))

Table 4.6: Comparative analysis of three models: multivariate volatility forecasting

Models	MSE	MAE	MAPE
Transformer Network(LSTM layer)	2.4181e-06	0.0016	51.9926
Transformer Network(CNN layer)	2.0115e-05	0.0056	92.1289
Transformer Network(CNN-LSTM layer)	2.7377e-06	0.0015	49.9926

S&P 500 volatility forecasting, were replicated exactly as in the previous part. Table 4.6 presents the results of performance comparison of the test data by employing three different layer configurations.

4.4 Comparision with LSTM and DDEWMA Models

In Chapters 2.2.1 and 2.2.2 we introduced these models. We used Time2Vec with LSTM network in order to compare its result with TN. By incorporating the time vector through Time2Vec, the LSTM model gains the ability to capture and utilize

Table 4.7: LSTM model performance - univariate volatility forecasting

Model	MSE	MAE	MAPE
LSTM	1.1678e-05	0.0021	28.0900

Table 4.8: LSTM model performance - multivariate volatility forecasting

Model	MSE	MAE	MAPE
LSTM	3.3587e-05	0.0041	57.7487

temporal patterns effectively. We implemented LSTM network in two steps. Initially, we input only the volatility feature into the Time2Vec with LSTM model, while in the second step, we incorporated all features. To ensure a fair assessment between the LSTM model and the TN model, we employed identical evaluation metrics as those used for the TN model. This methodology enabled us to effectively evaluate and compare the performance of both models. Table 4.7 demonstrates the results of implementing Time2Vec with LSTM model. By utilizing all features as input for our Time2Vec+LSTM model, we obtained the results presented in Table 4.8.

Second model is DDEWMA. The key in the function DDEWMA is to loop through alpha in order to select the alpha that generates the lowest MSE. Alpha represents the optimal smoothing constant. Table 4.9 presents the results of the three metrics, providing an overview of the model's performance.

Table 4.9: DDEWMA model performance - univariate volatility forecasting

Model	MSE	MAE	MAPE
DDEWMA	2.69093e-05	0.0037	56.3953

4.5 Transformer Network(TN)

We proposed a TN with a MHA mechanism architecture for the task of forecasting stock volatility. To enhance its capabilities in this research, we integrated three different variations of the feed forward layer—namely CNN, LSTM, and a hybrid approach combining both CNN-LSTM —instead of relying solely on a simple linear layer. For initial approach univariate volatility forecasting, we applied three different data for computing volatility. These data included using the Open price, Adj Close price, and the mean of open and Adj Close price. Their corresponding results are presented in Table 4.2,Table 4.3, and Table 4.4. Across all three cases, it is evident that using the hybrid (CNN-LSTM) architecture as feed forward layer in the TN outperforms other models. This is supported by its lowest MSE, MAE, and MAPE values. The LSTM-based feed forward layer also demonstrated competitive results, albeit with slightly higher MSE, MAE, and MAPE values. In contrast, the CNN-based feed forward layer variant showed higher MSE, MAE, and MAPE values, indicating relatively lower accuracy in capturing the true fluctuations over time. Regarding Table 4.4, none of the TN models with different feed forward layers performed well in capturing the volatility forecasting of the S&P index. Therefore, using the mean of the Open and Adj Close prices did not prove useful for accurate volatility forecasting. Computing volatility by taking the mean of the Open and Adj Close prices resulted in the loss of specific price dynamics that were crucial for volatility forecasting. Furthermore, it failed to adequately reflect important events or market changes occurring during the trading day. Based on the result of comparison obtained by using the Open price and Adj Close price led to compute volatility, we can conclude:

1. utilizing the Adj Close price is more suitable for capturing volatility patterns.

2. the hybrid model architecture incorporating both CNN and LSTM layers as feed forward layer is effective for volatility forecasting.

By closely analysing Figure 4.3 it becomes apparent that incorporating the Adj Close price for volatility forecasting adeptly captures the inherent fluctuations and patterns within the domain of volatility forecasting. The forecast values closely correspond to the actual values, exemplifying a remarkable precision in capturing the dynamics of volatility. While it is acknowledged that certain instances exist where the actual and forecast values do not align perfectly, a consistent pattern of upward and downward movements can be observed in both, particularly within the majority of the test data. This observation demonstrates the model's ability to effectively capture broader trends and shifts in volatility, thereby demonstrating its efficacy in forecasting volatility patterns.

The smooth and steadily decreasing MSE loss plots in Figure 4.2 and Figure 4.4, indicate a well-behaved training process where the model successfully optimizes its parameters to minimize forecasting errors. Analyzing the MSE loss plot alongside the comparison of actual and forecast volatility values allows for a more comprehensive assessment of the model's performance. By considering all the results, we can conclude that the hybrid (CNN-LSTM) feed forward layer within the TN outperforms other models in volatility forecasting. For the second approach multivariate volatility forecasting, based on the result of the MSE, MAE, and MAPE presented in Table 4.5 and Table 4.6, it becomes evident that employing the Adj Close price for volatility forecasting yields superior results in comparison to Open price. Furthermore, it's

worth noting that univariate volatility forecasting outperforms multivariate volatility forecasting.

To compare between the Time2Vec+LSTM model and the Time2Vec+TN model, we utilized identical evaluation metrics as employed for the TN model. The results demonstrated the strong performance of the Time2Vec+LSTM model on the dataset, as evidenced by the low values obtained for MSE, MAE, and MAPE when using only one feature. These metrics indicate that the model's forecasting align closely with the actual values, highlighting its accuracy. Furthermore, when comparing to the results achieved by the TN model using either LSTM or hybrid (CNN-LSTM) layers as the feed forward layer, the LSTM model performs competitively. The results achieved from these models exhibit a remarkable power, suggesting that the LSTM model performs on par with the TN model in terms of its ability. Figure 4.7 provides a comparison between the TN model and the LSTM model in terms of their ability to forecast volatility using a limited test dataset (zooming on 140 Data Points). The plot clearly illustrates that both models exhibit promising performance in capturing the ups and downs of volatility. They demonstrate a notable ability to track and forecast the fluctuations in volatility, showcasing their effectiveness in capturing the overall trends in the data. Based on the results presented in Table 4.3, Table 4.7, and Table 4.8, it can be concluded that the LSTM model demonstrates strong performance in univariate volatility forecasting comparable to the performance of the TN model. However, as the complexity of the dataset increases, along with the length of data and the number of features, the LSTM model shows poorer performance. Hence, it can be inferred that the TN model is better equipped to handle the complex dataset, longer

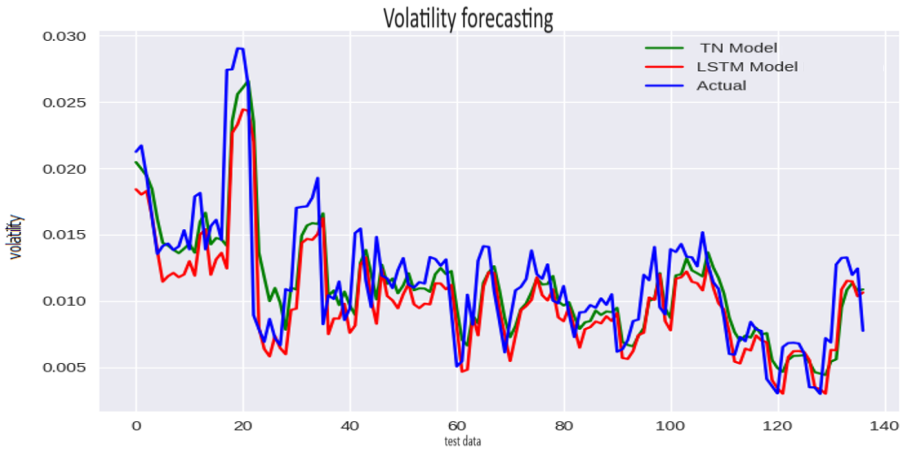

Figure 4.7: Comparing the performance of two models in univariate volatility forecasting

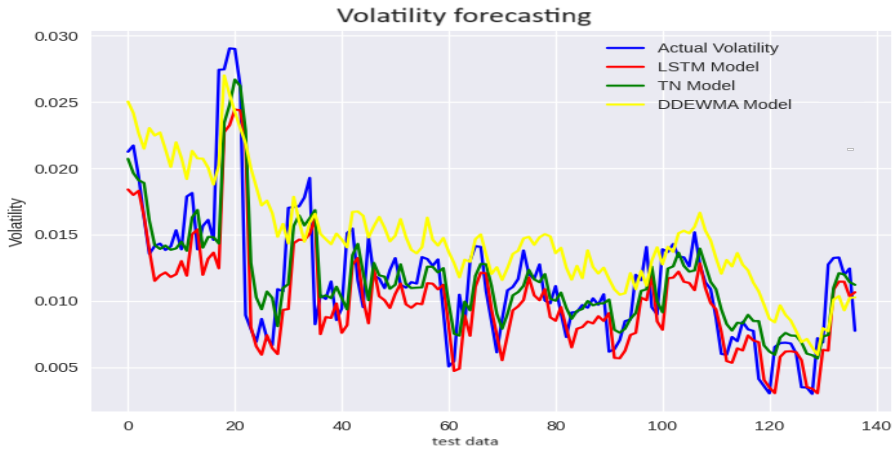

Figure 4.8: Comparing the performance of three models in univariate volatility forecasting

sequences of data, and multiple features, making it more suitable for stock volatility forecasting. Table 4.10 demonstrates that DDEWMA model showed unsatisfactory results in effectively capturing the volatility change of the S&P index as evidenced by its higher MAE and MAPE scores. We compared the results from TN, LSTM, and DDEWMA models in Figure 4.8 focusing on a limited test dataset (zooming on 140 data points). Table 4.10 presents the results of the three error metrics for these

models.

Table 4.10: Result of three models - univariate volatility forecasting

Model	MSE	MAE	MAPE
DDEWMA	2.69093e-05	0.0037	56.3953
LSTM	1.1678e-05	0.0021	28.0900
Transformer Network(CNN-LSTM layer)	1.0801e-05	0.0021	26.9638

In the next chapter we present our research plan focusing on the Informer model and its efficiency in handling LSTF tasks.

Chapter 5

Informer Network Model

5.1 Benefits of the Informer Model for Effective Sequence Forecasting: Evaluation of Execution Time and Performance Measures

In forecasting tasks, the concept of the impact of increasing the forecast length on computational complexity, memory utilization, and forecast accuracy are important. In the realm of sequence forecasting as mentioned in Chapter 2.2.4, the Informer model offers a promising solution to address the computational complexity and memory limitations associated with the basic TN architecture.

In this study, we investigate the benefits of the Informer model by analyzing its performance on a dataset comprising 17,750 data points across varying forecast lengths. Table 5.1 shows the data with different metrics and execution time for different forecast lengths by using Informer model. We used Informer model for forecasting, main-

Table 5.1: Analysis of execution time in Informer model for variable forecast lengths

Number of Data	Encoder Input Length	Decoder Input Length	Forecast Length	MSE	MAE	MAPE	Execution time (second)	Execution time (minute)
17750	336	48	12	2.71E-06	0.00102	0.3961	1695.9561	28.26
17750	336	48	24	2.71E-06	0.00103	0.3968	1742.2091	29.03
17750	336	48	48	2.71E-06	0.00111	0.3987	1786.7998	29.77
17750	336	48	72	3.37E-06	0.00114	0.3990	1902.9816	31.71
17750	336	48	96	3.50E-06	0.00117	0.4065	1914.6412	31.91
17750	336	48	168	3.64E-06	0.00119	0.4077	1990.0238	33.16
17750	336	48	216	3.65E-06	0.00120	0.4070	2263.3784	37.72
17750	336	48	336	3.65E-06	0.00120	0.4171	2607.3044	43.45
17750	336	48	480	3.73E-06	0.00121	0.4203	2984.1245	49.76

taining fixed encoder and decoder input lengths at 336 and 48 respectively. The forecast lengths were varied, ranging from 12 to 480-time steps. The dataset was divided into training, validation, and testing sets, and the model was trained for 20 epochs across all forecast lengths. Metrics including MSE, MAE, and MAPE were employed to assess forecasting performance. Additionally, the execution time of the model was recorded for each forecast length. Table 5.2 presents the hyperparameters of the Informer model. To compute execution time, we employed a simplified model across various forecast lengths while maintaining a record of their performance. It's important to note that augmenting factors such as increasing encoder and decoder layers, attention heads, model dimensions, batch size, dropout rates, and other factors, resulted in higher execution times.

The experiments were conducted using Google Colab equipped with a Tesla T4 GPU, an Intel Xeon CPU @ 2.20 GHz, 13 GB RAM, a Tesla K80 accelerator, and 12 GB GDDR5 VRAM.

Table 5.2: Hyperparameters of the Informer model

Parameter	**Value**	**Parameter**	**Value**
Input sequence length encoder	336	Time frame	minute-scale
Sequence length of decoder	48	Loss	MSE
Dimension of the model	512	Dropout	0.1
Number of encoder layers	2	Learning rate	0.0001
Number of decoder layers	1	Activation	ELU
Number of heads	16	Batch Size	32
Input feature	7	Epoch	20

5.2 Benefits of the Informer Model for Effective Sequence Forecasting

The comparison between forecast lengths 12 and 480 highlights the efficiency of the Informer model in addressing the challenges posed by forecasting long sequence time series data. The increase in execution time is moderate, growing from 28.26 minutes for a forecast length of 12 to 49.76 minutes for a forecast length of 480, equating to a factor of 1.7 (resulting in a ratio of $49.76/28.26 = 1.7$). This increase in execution time aligns with the notable growth in forecast length, which stands at a factor of 40 ($480/12 = 40$). This observation is consistent with the benefits derived from the ProbSparse attention mechanism and distilling operation integrated into the Informer model, which effectively manage computational complexity and memory usage.

The Informer model presented in Figure 5.1 shows impressive efficiency when handling diverse forecast lengths. It adeptly balances the trade-off between producing accurate

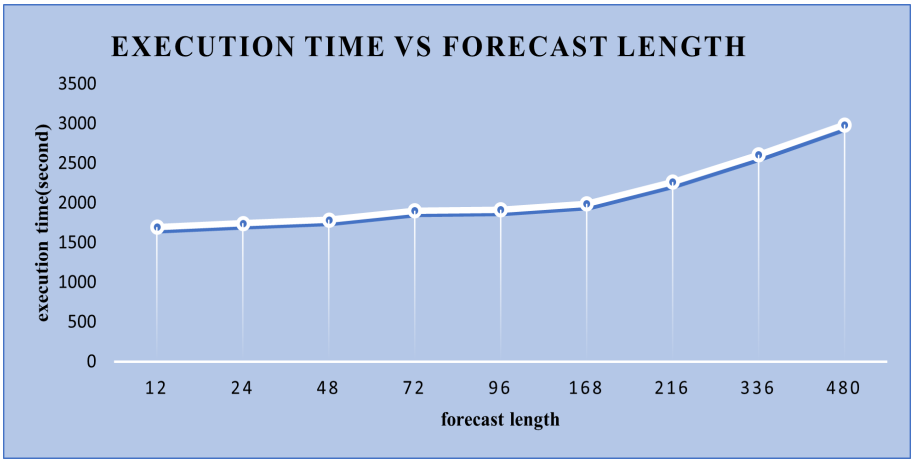

Figure 5.1: Execution time for varying forecast lengths

long-term forecasts and maintaining manageable execution times. This efficiency positions the Informer model as a promising option for time series forecasting tasks that involve extensive forecast horizons. It provides tangible advantages without compromising on accuracy.

5.2.1 Understanding Statistical Error Across Different Forecast Lengths

The Informer model shows adaptable performance in terms of MSE, MAE, and MAPE across different forecast lengths. However, it's crucial to observe the proportional rise in execution time as the forecast length is increased, yet this increase is not so huge. This balance between accuracy and execution time underscores the model's efficiency in handling diverse time series forecasting scenarios. The MSE values provide insight into the accuracy of the forecasting model. Lower MSE values are desirable as they indicate closer alignment between forecast and actual values. As

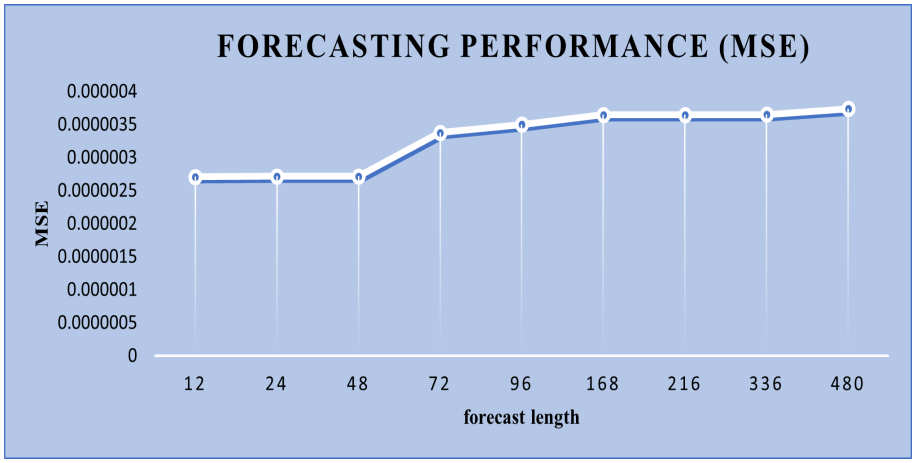

Figure 5.2: Forecasting performance (MSE) of Informer model with different forecast length

shown in Figure 5.2, MSE consistently increases but moderately as the forecast length extends from 12 to 480 units. This suggests that the model's forecast accuracy tends to decrease for longer forecast horizons. The observed trend of increasing MSE with longer forecast lengths could be attributed to the inherent complexity of forecasting tasks over extended periods.

Figure 5.3 demonstrates that the model's accuracy, as measured by MAE, experiences a modest increase with longer forecast lengths. This trade-off between forecast accuracy and forecast lengths should be considered when using the model for different forecasting tasks.

The MAPE trends across different forecast lengths reveal that the model's forecast accuracy is relatively stable for a wide range of forecast horizons can be seen in Figure 5.4. While there is a slight increase in MAPE for longer forecast lengths, this is consistent with the challenges posed by long-range forecasting. These findings highlight the model's effectiveness in providing accurate forecasting for various time

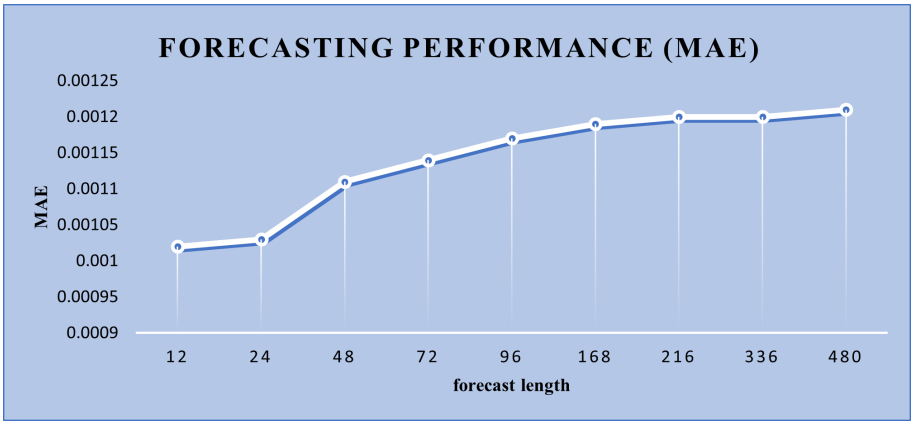

Figure 5.3: Forecasting performance (MAE) of Informer model with different forecast length

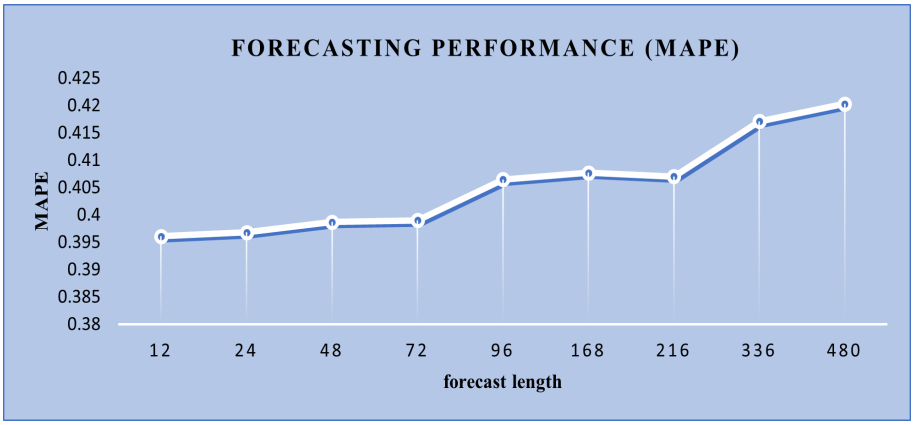

Figure 5.4: Forecasting performance (MAPE) of Informer model with different forecast length

horizons, allowing for informed decision-making across different planning scenarios.

In the next chapter we summarize our research findings and discuss future directions for our work.

Chapter 6

Conclusions and Future Work

In this research, we proposed a TN with a MHA mechanism for forecasting stock volatility. To improve the model's performance, we incorporated three variations of the feed forward layer: CNN, LSTM, and a hybrid approach combining both (CNN-LSTM). We conducted experiments using different features for computing volatility, including the Open price and the Adj Close price. The results showed that the TN with the hybrid CNN-LSTM layer as feed forward layer outperformed other models. When comparing the result of univariate volatility forecasting versus multivariate volatility forecasting, it was found that employing the Adj Close price alone yielded superior results for forecasting the volatility of the S&P 500 index in both of them. Furthermore, it is important to highlight that the utilization of univariate volatility forecasting surpasses the performance of multivariate volatility forecasting.

In the first comparison approach, we compared three models: DDEWMA, the Time2Vec with LSTM model, and the Time2Vec with TN model. We focused on using only one feature for volatility forecasting. Based on the results, both the LSTM and TN mod-

els showed a significant ability to forecast volatility. However, the DDEWMA model performed comparatively worse than the others.

In the second comparison approach, we again compared the Time2Vec+LSTM model and the Time2Vec+TN model, but this time we considered all seven features. While the Time2Vec+LSTM model exhibited a remarkable ability to track and forecast volatility fluctuations using a single volatility feature, its performance declined when multiple features were involved. On the other hand, the TN model with MHA proved to be more suitable for handling complex datasets and multiple features. Therefore, in scenarios involving stock volatility forecasting with various features, the TN model with MHA is the preferred choice.

In the realm of sequence forecasting, the Informer model offers a promising solution to address the computational complexity and memory limitations associated with the basic TN architecture. This study investigated the benefits of the Informer model by analyzing its performance on a Bitcoin minute-scale dataset across varying forecast lengths. Based on the results, the advantages of the Informer model, namely the ProbSparse attention mechanism and the distilling operation, significantly contributed to its efficiency in handling LSTF tasks. Result of MSE, MAE, and MAPE showed that the when the forecast horizon is short, the Informer model consistently maintains stable forecasting performance. However, as the forecast duration progressively extended, the performance of each model's was affected. Our analysis of the provided dataset affirms that the execution time of Informer grew moderately with increased forecast length, indicating its robustness in addressing quadratic computational complexity and memory bottlenecks. These advantages collectively emphasize

the practical benefits of the Informer model in achieving accurate forecasts for extended forecast horizons while maintaining computational efficiency. It is important to acknowledge that these observations pertain specifically to the dataset and model configuration considered in this study. However, it is essential to note that the results from TN, as used in our research, cannot be directly applied to real-market forecasting without adjustments. This is because factors such as transaction cost, fees and other market dynamics, which significantly impact outcomes, are not included in our model. These omissions can affect the accuracy of forecasting when applied to live trading scenarios.

There are several ways as listed below to further enhance our understanding and improve the models' performance and are left as future work.

- Explore further modifications and enhancements to the TN architecture with different hyperparameters, including different encode-decoder layers, different number of heads in the multi-head attention mechanism.

- Investigate the impact of using different data sources, including alternative financial instruments (e.g., cryptocurrencies).

- Investigate methods for integrating multi-modal data sources into the TN model. This could include combining time series data with textual data.

- Investigate hybrid models that combine TN models with other DL architectures such as RNN-based models.

- Implement LSTF-based volatility forecasting with the Informer model incorporating different data sources such as stocks and indexes and consider diverse

timeframes.

Publications

1. Golnaz Sababipour Asl, Ruppa Thulasiram, and Aerambamoorthy Thavaneswarn, *Stock Volatility Forecasting with Transformer Network*, In proceedings of the 2023 IEEE Symposium on Computational Intelligence in Financial Engineering and Economics (CIFEr), pp.90-96, Dec. 2023 (held in conjunction with IEEE Computational Intelligence Society Symposium Series on Computational Intelligence). (Nominated for Best Paper award - will be presented on Dec.6, 23)

2. *Stock Volatility Forecasting with Transformer and Informer Networks* - in preparation for submission to a relevant journal.

Bibliography

[1] Ashish Vaswani, Noam M. Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *ArXiv*, abs/1706.03762, 2017.

[2] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. Informer: Beyond efficient transformer for long sequence time-series forecasting. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 11106–11115, 2021.

[3] Cong Xu, Huiling Huang, Xiaoting Ying, Jianliang Gao, Zhao Li, Peng Zhang, Jie Xiao, Jiarun Zhang, and Jiangjian Luo. Hgnn: Hierarchical graph neural network for predicting the classification of price-limit-hitting stocks. *Information Sciences*, 607:783–798, 2022.

[4] Zhang Xiao-Wen and Zeng Min. An empirical study on big data stock volatility forecasting algorithm based on multivariate hybrid criterion fuzzy model. *The International Journal of Electrical Engineering & Education*, 0(0):0020720920983995, 2021.

[5] Zimo Zhu, Aerambamoorthy Thavaneswaran, Alexander Paseka, Julieta Frank,

and Ruppa Thulasiram. Portfolio optimization using a novel data-driven ewma covariance model with big data. In *2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)*, pages 1308–1313, 2020.

[6] Tim Bollerslev. Generalized autoregressive conditional heteroskedasticity. *Journal of Econometrics*, 31(3):307–327, 1986.

[7] Robert F Engle, Clive WJ Granger, and Dennis Kraft. Combining competing forecasts of inflation using a bivariate arch model. *Journal of economic dynamics and control*, 8(2):151–165, 1984.

[8] Valeria D'Amato, Susanna Levantesi, and Gabriella Piscopo. Deep learning in predicting cryptocurrency volatility. *Physica A: Statistical Mechanics and its Applications*, 596:127158, 2022.

[9] Chaojie Wang, Yuanyuan Chen, Shuqi Zhang, and Qiuhui Zhang. Stock market index prediction using deep transformer model. *Expert Systems with Applications*, 208:118128, 2022.

[10] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning Representations by Back-propagating Errors. *Nature*, 323(6088):533–536, 1986.

[11] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[12] Yang Liu. Novel volatility forecasting using deep learning–long short term memory recurrent neural networks. *Expert Systems with Applications*, 132:99–109, 2019.

[13] Hualing Lin and Qiubi Sun. Financial volatility forecasting: A sparse multi-head attention neural network. *Information*, 12(10), 2021.

[14] Dung Hoang Anh Mai, Linh Thanh Nguyen, and Eun Yeol Lee. Tssnote-cyaprombert: Development of an integrated platform for highly accurate promoter prediction and visualization of synechococcus sp. and synechocystis sp. through a state-of-the-art natural language processing model bert. *Frontiers in Genetics*, 13:1067562, 2022.

[15] Abdullatif Köksal and Arzucan Özgür. Twitter dataset and evaluation of transformers for turkish sentiment analysis. In *2021 29th Signal Processing and Communications Applications Conference (SIU)*, pages 1–4, 2021.

[16] Tomáš Výrost Štefan Lyócsa, Peter Molnár. Stock market volatility forecasting: Do we need high-frequency data? *International Journal of Forecasting*, 37(3):1092–1110, 2021.

[17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[18] Fuli Feng, Huimin Chen, Xiangnan He, Ji Ding, Maosong Sun, and Tat-Seng Chua. Enhancing stock movement prediction with adversarial training. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 5843–5849. International Joint Conferences on Artificial Intelligence Organization, 7 2019.

[19] TN Kipf and M Welling. Semi-supervised classification with graph convolutional networks. international conference on learning representations, 2017.

[20] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *stat*, 1050(20):10–48550, 2017.

[21] Weiwei Jiang. Applications of deep learning in stock market prediction: Recent progress. *Expert Systems with Applications*, 184:115537, 2021.

[22] Yuze Lu, Hailong Zhang, and Qiwen Guo. Stock and market index prediction using informer network. *arXiv preprint arXiv:2305.14382*, 2023.

[23] Aerambamoorthy Thavaneswaran, Alex Paseka, and Julieta Frank. Generalized value at risk forecasting. *Communications in Statistics-Theory and Methods*, 49(20):4988–4995, 2020.

[24] Sen Jaydip Mehtab, Sidra. Stock price prediction using cnn and lstm-based deep learning models. In *2020 International Conference on Decision Aid Sciences and Application (DASA)*, pages 447–453. IEEE, 2020.

[25] Schmidhuber Jürgen Hochreiter, Sepp. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[26] Andrew R Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information theory*, 39(3):930–945, 1993.

[27] Jose Castro, Pedro Achanccaray Diaz, Ieda Sanches, Laura Cue La Rosa, Patrick

Nigri Happ, and Raul Feitosa. Evaluation of recurrent neural networks for crop recognition from multitemporal remote sensing images. 11 2017.

[28] Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111, Doha, Qatar, October 2014. Association for Computational Linguistics.

[29] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv*, 2014.

[30] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[31] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[32] SGOPAL Patro and Kishore Kumar Sahu. Normalization: A preprocessing stage. *arXiv preprint arXiv:1503.06462*, 2015.

[33] Sashank Sridhar and Sowmya Sanagavarapu. Multi-head self-attention transformer for dogecoin price prediction. In *2021 14th International Conference on Human System Interaction (HSI)*, pages 1–6. IEEE, 2021.

[34] Feng Zhou, Qun Zhang, Yuan Zhu, and Tian Li. T2v_tf: An adaptive timing

encoding mechanism based transformer with multi-source heterogeneous information fusion for portfolio management: A case of the chinese a50 stocks. *Expert Systems with Applications*, 213:119020, 2023.

[35] Seyed Mehran Kazemi, Rishab Goel, Sepehr Eghbali, Janahan Ramanan, Jaspreet Sahota, Sanjay Thakur, Stella Wu, Cathal Smyth, Pascal Poupart, and Marcus Brubaker. Time2vec: Learning a vector representation of time. *arXiv preprint arXiv:1907.05321*, 2019.

[36] Harshit Kumar Lohani, S Dhanalakshmi, and V Hemalatha. Performance analysis of extreme learning machine variants with varying intermediate nodes and different activation functions. In *Cognitive Informatics and Soft Computing: Proceeding of CISC 2017*, pages 613–623. Springer, 2019.

[37] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[38] Cort J. Willmott and Kenji Matsuura. Advantages of the mean absolute error (mae) over the root mean square error (rmse) in assessing average model performance. *Climate Research*, 30(1):79–82, 2005.

[39] Arnaud de Myttenaere, Boris Golden, Bénédicte Le Grand, and Fabrice Rossi. Mean absolute percentage error for regression models. *Neurocomputing*, 192:38–48, jun 2016.

[40] Rob J. Hyndman. *Moving Averages*, pages 866–869. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.