

TDK-thesis

Nguyen Duy Chien

Bui Nguyen Kim Hai

Transformer Encoder and Multi-features Time2Vec for Financial Prediction

EÖTVÖS LORÁND UNIVERSITY

FACULTY OF INFORMATICS

DEPARTMENT OF NUMERICAL ANALYSIS



Authors:

Nguyen Duy Chien

Computer Science BSc

III. grade

Bui Nguyen Kim Hai

Computer Science BSc

III. grade

Supervisor:

Dr. Gergő Bognár

Assistant Professor

Budapest, Dec 2024

Abstract

Financial prediction is a complex task, influenced by a multitude of factors that affect stock prices. Specifically, financial time series data present a significant challenge for prediction due to the complexity involved in modeling both short-term fluctuations and long-term temporal dependencies within the dataset. Transformers have remarkable success in many tasks in natural language processing and computer vision using attention mechanisms, which also triggered great interest in the time series community. The ability to capture long-range dependencies and interaction and the ability to learn and understand languages lead to the understanding of the financial market and recognize the historical pattern of prices. Although the application of Transformer models in stock prediction is well-established, existing research predominantly emphasizes the use of historical prices of individual features for singular predictions. This narrow focus constrains the model's ability to capture and understand broader market trends. In particular, within sectors such as finance and technology, energy companies belonging to the same industry often exhibit correlated stock price movements.

In this thesis, we developed a novel neural network architecture; we integrated Time2Vec with the Encoder of the Transformer model. Based on the study of different markets, we proposed a novel correlation feature selection method. Through a comprehensive fine-tuning process involving multiple hyperparameters, we conducted a comparative analysis of our results against several benchmark models. Based on this analysis, we identified key insights and proposed potential avenues for further enhancing the performance and applicability of our approach in future research. We conclude that our method outperforms other state-of-the-art encoding methods such as positional encoding, and we also conclude that selecting correlation features enhance the accuracy of predicting multiple stock prices.

— ACM Classification —

- Computing methodologies / Machine learning
- Applied computing / Economics
- Computing methodologies / Model development and analysis

Contents

1	Introduction	2
2	Background	4
2.1	Mathematical Models	4
2.2	Deep Learning with Neural Networks	5
2.3	Optimizers	8
2.4	Evaluation Metrics	10
2.5	Time2Vec	12
2.6	Transformer	13
2.7	Geometric Mean Not NaN	15
2.8	Correlation Analysis	16
3	TT2VFin: Transformer Time2Vec Financial	18
3.1	Motivation	18
3.2	Methodology	19
3.3	Data	21
3.4	Proposed Model	25
3.5	Hyperparameters Tuning	28
4	Result and Evaluation	30
4.1	Comparing to SOTA models	31
4.2	Predicting Trends and Prices	33
5	Summary	40
	Bibliography	42
	List of Figures	46
	List of Tables	47

Chapter 1

Introduction

Time series forecasting is challenging, especially in the financial industry [1]. It involves statistically understanding complex linear, and non-linear interactions within historical data to predict the future. Traditional statistical approaches commonly adopt linear regression, exponential smoothing [2], and auto regression model [3]. With the advances in deep learning, recent works are heavily invested in ensemble models and sequence-to-sequence modeling such as RNN (Recurrent Neural Networks) and LSTM (Long Short-Term Memory) [4]. However, the primary drawback of these methods is that the RNN family struggles to capture extremely long-term dependencies [5]. A well-known sequence-to-sequence model called Transformer [6] has achieved great success in NLP (Natural Language Processing), especially LLMs (Large Language Models) like ChatGPT and Gemini. Different from RNN-based models, the Transformer employs a multi-head self-attention mechanism to learn the relationship among different positions globally.

Stock Forecasting

Stock Market is a system where currency, stocks, equities, and other financial commodities are either bought or sold. The stock market also allows people to own some shares of public companies [7]. Traders generally prefer to purchase stocks of companies that show potential for future growth and avoid those expected to decline in value. The most commonly used methods for predicting stock returns are broadly categorized into fundamental analysis and technical analysis. Fundamental analysis is particularly essential for investors focused on long-term trading.

Therefore, in this world of financial markets, it is a demand to forecast the stock price using statistical analysis [8], deep learning, and machine learning techniques to maximize the capital gain and decrease the loss.

Predicting a fast and accurate model for stock price forecasting is a challenging task, and this is an active area of research where it is yet to be found which is the best way to forecast the stock price [7].

Motivated by the success of deep learning in image classification, natural language processing, and time series analysis, deep learning has become an emerging approach in the field of finance as well. Deep neural networks automatically find the corresponding representation of low dimension by extracting high-dimensional input data. Machine and deep learning techniques have been found to be efficient in price forecasting and market trend prediction, see e.g. [9], that support investors to anticipate future trends and optimize stock investments.

The structure of this document is organized as follows:

- Chapter 2 presents a comprehensive overview of time series forecasting methods. This includes an exploration of traditional mathematical modeling techniques, such as statistical and probabilistic approaches, as well as modern advancements in deep learning architectures, highlighting their strengths and limitations in different forecasting scenarios.
- Chapter 3 outlines the motivation behind our proposed model, detailing the specific challenges it seeks to address. This chapter also provides a thorough explanation of the methodology, including the design principles, architectural choices, and underlying theoretical framework that drive the development of the model.
- Chapter 4 presents a comprehensive empirical analysis of the proposed model. It details the experimental methodology, datasets, and performance metrics, offering a systematic comparative evaluation against state-of-the-art models.
- Chapter 5 provides a concise summary of our work, highlighting its key contributions and outlining potential future directions that can build upon the foundations established by this research.

Chapter 2

Background

In this chapter, we overview the most important related techniques in the field and introduce the deep learning methodologies that we utilized for model development.

2.1 Mathematical Models

Traditional forecasting techniques utilize mathematical tools based on statistical analysis. Here, we only provide a brief overview of the most widely used methods.

ARIMA

Typical forecasting techniques in the literature utilize statistical tools, such as ETS (Exponential Smoothing) and ARIMA (Auto Regressive Integrated Moving Average) [2, 3], on numerical time series data for making one-step-ahead predictions. These predictions are then recursively fed into the future inputs to obtain multi-step forecasts. Multi-horizon forecasting methods [10, 11] directly generate simultaneous predictions for multiple predefined future time steps.

SARIMA

The SARIMA (Seasonal Autoregressive Integrated Moving Average) model is an extension of the ARIMA model that incorporates seasonal components to better handle time series data with recurring patterns. SARIMA includes additional terms for SAR (seasonal autoregression), SI (seasonal differencing), and SMA (seasonal moving averages), along-

side the standard AR, I, and MA components [12]. This allows SARIMA to capture both non-seasonal and seasonal dependencies.

2.2 Deep Learning with Neural Networks

In this section, we overview some deep learning techniques relevant to our work.

DNNs (Deep Neural Networks)

A neural network (also artificial neural network or neural net, abbreviated ANN or NN) is a model inspired by the structure and function of biological neural networks in animal and human brains.

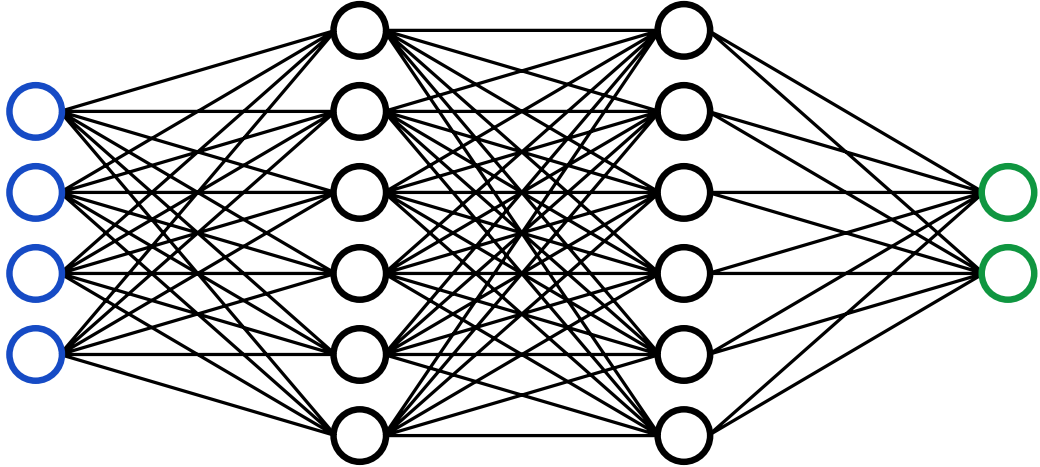


Figure 2.1: A schematic feedforward neural network with an input, two hidden, and an output layer [13].

Neural networks consist of neurons organized into layers (Figure 2.1), where the layers usually perform a linear transformation followed by a non-linear activation function. This allows networks to learn complex patterns and relationships in the data. There are numerous different activation functions out there, but we use **ReLU** in this study. **ReLU** (**R**ectified **L**inear **U**nit) is a popular activation function in neural networks, defined as:

$$\text{ReLU}(x) = \max(0, x) \quad (2.1)$$

It introduces non-linearity by setting all negative values to zero while leaving positive values unchanged. This simple approach helps networks learn faster and reduces issues with vanishing gradients, making it particularly effective in deep learning models [14].

RNNs (Recurrent Neural Networks)

RNNs are designed to handle temporal problems, and they are widely used for sequential data and time-series analysis. The RNN family is a natural choice for financial forecasting as well [15]. Although RNN is able to accurately characterize the contextual relationship between sequential data, this relationship weakens as the gap distance between them grows. Figure 2.2 presents the unfolded architecture of an RNN.

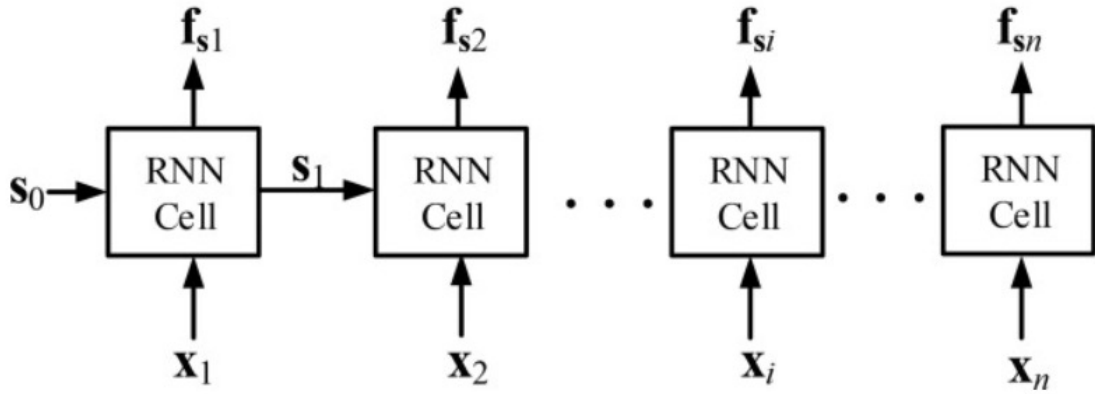


Figure 2.2: A schematic RNN unfolded to layers [16].

Moreover, an RNN model has the vanishing gradient problem for the long sequence data. However, LSTM can prevent this problem during training.

CNNs (Convolutional Neural Networks)

CNNs involve special layers that perform convolution and pooling, motivated by the behavior of the human visual cortex. CNN architectures serve as the most widely used ANNs in image processing and computer vision that perform representation learning, i.e., a dimension reduction of the input data and the automatic extraction of the important features. One of the famous architectures that successfully applies CNN is LeNet (Figure 2.3).

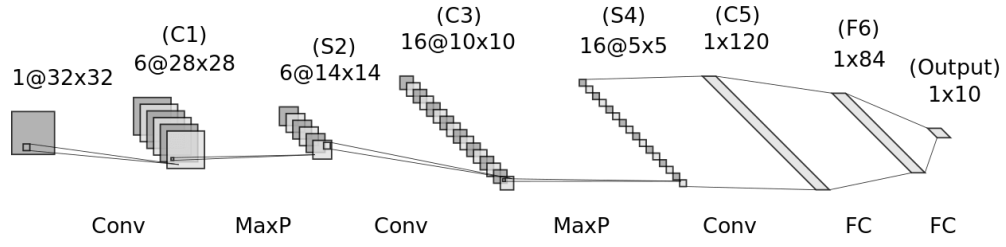


Figure 2.3: LeNet architecture: Conv (convolutional) and MaxP (maximum pooling) layers are combined with FC (fully-connected feedforward layers) [17].

LSTM (Long Short-Term Memory)

LSTM is a special RNN that has a memory mechanism with gates, making the network capable of learning long-term dependencies and preventing the vanishing gradient problem. The structure of an LSTM cell is demonstrated in Figure 2.4.

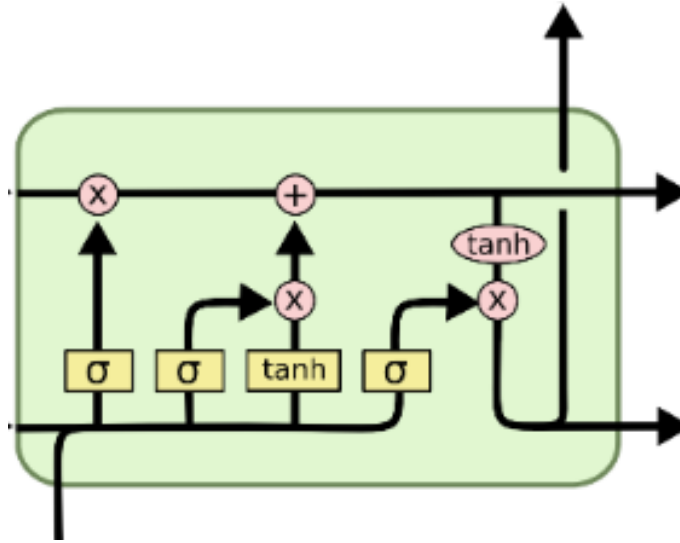


Figure 2.4: LSTM cell structure with hidden states and gates [18].

LSTMs [4] are widely used for financial time series prediction, particularly in forecasting stock prices. There are also many variables of LSTMs, such as BiLSTMs and Conv-LSTMs [19], or Attentive-LSTMs [20] with an attention mechanism to predict stock price movement. However, Shiyang Li et al. point out that LSTM can only distinguish 50 positions nearby with an effective context size of about 200 [5]. That means that LSTM-based models suffer from the difficulty in capturing extremely long-term dependencies in time series.

2.3 Optimizers

In supervised learning, the training of ANNs is usually carried out as a data-driven numerical optimization of a loss function $\mathcal{L}$ that has the form of a sum:

$$\mathcal{L}(\omega) = \frac{1}{n} \cdot \sum_{i=1}^n \mathcal{L}_i(\omega),$$

where the parameter ω that minimizes $\mathcal{L}(\omega)$ is to be estimated. Each summand function $\mathcal{L}_i(\omega)$ is typically associated with the i^{th} observation in the data set (used for training). The parameter then be updated with:

$$\omega := \omega - \eta \cdot \nabla \mathcal{L}(\omega) = \omega - \frac{\eta}{n} \cdot \sum_{i=1}^n \nabla \mathcal{L}_i(\omega),$$

where η is a learning rate. In the following, we overview the most widely used optimizers in deep learning that we studied during the model development.

SGD (Stochastic Gradient Descent)

The paper titled *A Stochastic Approximation Method* [21] by Robbins and Monro is foundational for SGD. This work formalized the concept of stochastic approximation, which is central to SGD's use of noisy or mini-batch updates.

$$\omega := \omega - \eta \cdot \nabla \mathcal{L}_i(\omega).$$

Pros: Simple and computationally efficient.

Cons: Prone to oscillations, slow convergence in regions of low gradient or high curvature, sensitive to learning rate.

Momentum

Momentum by Rumelhart, Hinton, and Williams [22] is an extension to the gradient descent optimization algorithm that builds inertia in a search direction to overcome local minima and oscillation of noisy gradients. It is based on the same concept of momentum in physics. This algorithm is used to accelerate the gradient descent algorithm by taking into consideration the ‘exponentially weighted average’ of the gradients. Using averages makes the algorithm converge towards the minima at a faster pace.

$$\Delta\omega := \alpha \cdot \Delta\omega - \eta \cdot \nabla \mathcal{L}_i(\omega),$$

$$\omega := \omega - \eta \cdot \nabla \mathcal{L}_i(\omega) + \alpha \cdot \Delta \omega,$$

where α is an exponential decay factor between 0 and 1.

Pros: Reduces oscillations, accelerates convergence in consistent gradient directions, helps avoid shallow local minima.

Cons: Requires manual learning rate tuning, may struggle with adaptive rate needs.

AdaGrad (Adaptive Gradient)

AdaGrad is a modified stochastic gradient descent algorithm with a per-parameter learning rate, first published in 2011 [23]. It still has a base learning rate η , but this is multiplied with the elements of a vector $\{G_{j,j}\}$, which is the diagonal of the outer product matrix.

$$\begin{aligned} g_\tau &= \nabla \mathcal{L}_i(\omega), \\ G &= \sum_{\tau=1}^t g_\tau \cdot g_\tau^T, \\ G_{j,j} &= \sum_{\tau=1}^t g_{\tau,j}^2, \\ \omega &:= \omega - \eta \cdot \text{diag}(G)^{-\frac{1}{2}} \odot g. \end{aligned}$$

Pros: Adapts learning rate for each parameter based on its history of gradients, beneficial for sparse data and NLP.

Cons: Learning rate diminishes too quickly, which can slow down training over time, especially for deep networks.

RMSProp (Root Mean Square Propagation)

RMSProp is an unpublished algorithm first proposed in the Coursera course. *Neural Network for Machine Learning* lecture 6 by Geoff Hinton. It tries to improve AdaGrad by using the exponential moving average instead of taking the cumulative sum of squared gradients like in AdaGrad.

$$\begin{aligned} v_\omega &:= \gamma \cdot v_\omega + (1 - \gamma) \cdot (\nabla \mathcal{L}_i(\omega))^2, \\ \omega &:= \omega - \frac{\eta}{\sqrt{v_\omega}} \cdot \nabla \mathcal{L}_i(\omega), \end{aligned}$$

where γ is a forgetting factor.

Pros: Solves AdaGrad’s diminishing learning rate issue by using an exponentially decaying average of squared gradients, leading to a more stable adaptive learning rate.

Cons: Still may need manual tuning of hyperparameters, though more stable than earlier methods.

Adam (Adaptive Moment Estimation)

Adam is an algorithm for an optimization technique for gradient descent. The method is really efficient when working with large problems involving a lot of data or parameters. It requires less memory and is efficient. Intuitively, it is a combination of the Momentum algorithm and the RMSProp algorithm. Adam inherits the strengths, or the positive attributes, of the two methods and builds upon them to give a more optimized gradient descent.

$$\begin{aligned} m_{\omega} &:= \gamma_1 \cdot m_{\omega} + (1 - \gamma_1) \cdot \nabla \mathcal{L}_i(\omega), \\ v_{\omega} &:= \gamma_2 \cdot v_{\omega} + (1 - \gamma_2) \cdot (\nabla \mathcal{L}_i(\omega))^2, \\ \hat{m}_{\omega} &= \frac{m_{\omega}}{1 - \gamma_1}, \\ \hat{v}_{\omega} &= \frac{v_{\omega}}{1 - \gamma_2}, \\ \omega &:= \omega - \eta \cdot \frac{\hat{m}_{\omega}}{\sqrt{\hat{v}_{\omega} + \varepsilon}}, \end{aligned}$$

where ε is a small scalar (e.g. 10^{-8}) used to prevent division by 0. γ_1 (e.g. 0.9) and γ_2 (e.g. 0.999) are the forgetting factors for gradients and second moments of gradients, respectively. Squaring and square-rooting are done element-wise.

Pros: Combines both Momentum and RMSProp (by tracking both the first moment, or mean, and the second moment, or variance, of the gradients), leading to adaptive learning rates and fast, stable convergence.

Cons: More hyperparameters to tune, sometimes less effectively if not tuned.

2.4 Evaluation Metrics

The trained models are usually evaluated by means of statistical metrics. In this section we overview the most important metrics that are going to be used in the thesis.

MAPE (Mean Absolute Percentage Error)

Also known as MAPD (Mean Absolute Percentage Deviation), it is a measure of prediction accuracy of a forecasting method in statistics. It usually expresses the accuracy as a ratio defined by the formula:

$$\text{MAPE} = 100 \cdot \frac{1}{n} \cdot \sum_{i=1}^n \left| \frac{Y_i - \hat{Y}_i}{Y_i} \right|, \quad (2.2)$$

where Y_i is the actual value, $\hat{Y}_i$ is the predicted value at i , and n is the number of values.

MAE (Mean Absolute Error)

MAE is a measure of errors between paired observations expressing the same phenomenon.

$$\text{MAE} = \frac{1}{n} \cdot \sum_{i=1}^n |Y_i - \hat{Y}_i|, \quad (2.3)$$

where Y_i is the actual value, $\hat{Y}_i$ is the predicted value at i , and n is the number of values.

RMSE (Root Mean Square Error)

RMSE is the residuals' standard deviation, or the average difference between the projected and actual values produced by a statistical model.

$$\text{RMSE} = \sqrt{\sum_{i=1}^n \frac{(Y_i - \hat{Y}_i)^2}{n}}, \quad (2.4)$$

where Y_i is the actual value, $\hat{Y}_i$ is the predicted value at i , and n is the number of values.

MSE (Mean Square Error)

MSE is the average squared difference between the value observed in a statistical study and the values predicted from a model.

$$\text{MSE} = \frac{1}{n} \cdot \sum_{i=1}^n (Y_i - \hat{Y}_i)^2, \quad (2.5)$$

where Y_i is the actual value, $\hat{Y}_i$ is the predicted value at i , and n is the number of values.

R2 (R-square)

R-squared value shows how well the model predicts the outcome of the dependent variable. R-squared values range from 0 to 1. Squared value of 0 means that the model explains or predicts 0% of the relationship between the dependent and independent variables.

$$R^2 = 1 - \frac{\sum_{i=1}^n (Y_i - \hat{Y}_i)^2}{\sum_{i=1}^n (Y_i - \bar{Y})^2}, \quad (2.6)$$

where Y_i is the actual value, $\hat{Y}_i$ is the predicted value at i , $\bar{Y}$ is the mean of all values, and n is the number of values.

2.5 Time2Vec

Time2Vec [24] is an approach providing a model-agnostic vector representation for time. Time decomposition technique that encodes a temporal signal into a set of frequencies. For a given scalar notion of time τ , Time2Vec of τ , denoted as $\mathbf{t2v}(\tau)$, is a vector of size $k + 1$ defined as follows:

$$\mathbf{t2v}(\tau)[i] = \begin{cases} \omega_i \cdot \tau + \phi_i & \text{if } i = 0 \\ \mathcal{F}(\omega_i \cdot \tau + \phi_i) & \text{if } 1 \leq i \leq k \end{cases} \quad (2.7)$$

where $\mathbf{t2v}(\tau)[i]$ is the i^{th} element of $\mathbf{t2v}(\tau)$, $\mathcal{F}$ is a periodic activation function, and ω_i and ϕ_i are learnable parameters. This equation has three important properties: capturing both periodic and non-periodic patterns, being invariant to time rescaling, and being simple enough to be combined with many models.

Why Time2Vec?

The $\mathcal{F}$ periodical activation function (sinusoidal functions) is inspired by parts of positional encoding from the paper *Attention Is All You Need* [6], which can be combined with transformers architecture. Based on the explanation of Kazemi et al. [24], Time2Vec serves as an alternative to positional encoding that can also capture periodic behavior not supported by positional encoding.

Positional Encoding: In Transformers, positional encoding is added to the input vectors to help the model capture the order of the input sequence. Typically, positional encoding is fixed and not learned during training. This fixed encoding uses sine and cosine functions to represent the positions in the sequence, allowing the model to be aware of the order without any recurrence or convolution.

Time2Vec: In contrast, Time2Vec uses sinusoidal functions to capture temporal or periodic patterns in data, but instead of using predefined frequencies, it learns the frequencies and phase shifts during training. This adaptability allows the model to capture patterns in time more accurately and flexibly, making it especially useful for stock prediction tasks where periodicity or temporal dynamics are critical – patterns might vary over time.

Generally, Time2Vec acts as a learnable positional encoding block. This conclusion means that we can use Time2Vec instead of normal position encoding before feeding the data into the Encoders.

2.6 Transformer

Attention mechanism

To address the bottleneck caused by fixed-length encoding vectors, where the decoder’s access to input information is limited – Vaswani et al. introduced the Transformer architecture to leverage the attention mechanism in *Attention Is All You Need* [6]. This approach overcomes the limitations of fixed-length representations, especially when handling long or complex sequences. In fixed-length encoding, the representation’s dimensionality must align with that of shorter or simpler sequences, making it difficult to capture all the necessary details in more complex cases. The attention mechanism allows the model to dynamically focus on different parts of the input sequence, improving the handling of intricate, long sequences without losing essential information.

Transformer

The innovation of the Transformer in deep learning has brought great interest recently due to its excellent performances in NLP [25]. Over the past few years, numerous Transformers have been proposed to advance the state-of-the-art performances of various tasks. Transformers have shown “significantly” great modeling ability for long-range

dependencies and interactions in sequential data and thus are appealing to time series modeling. Many variants of Transformer have been proposed to address special challenges in time series modeling and have been successfully applied to various time series tasks, such as forecasting, anomaly detection, and classification [26]. Figure 2.5 shows the architecture of the Transformer.

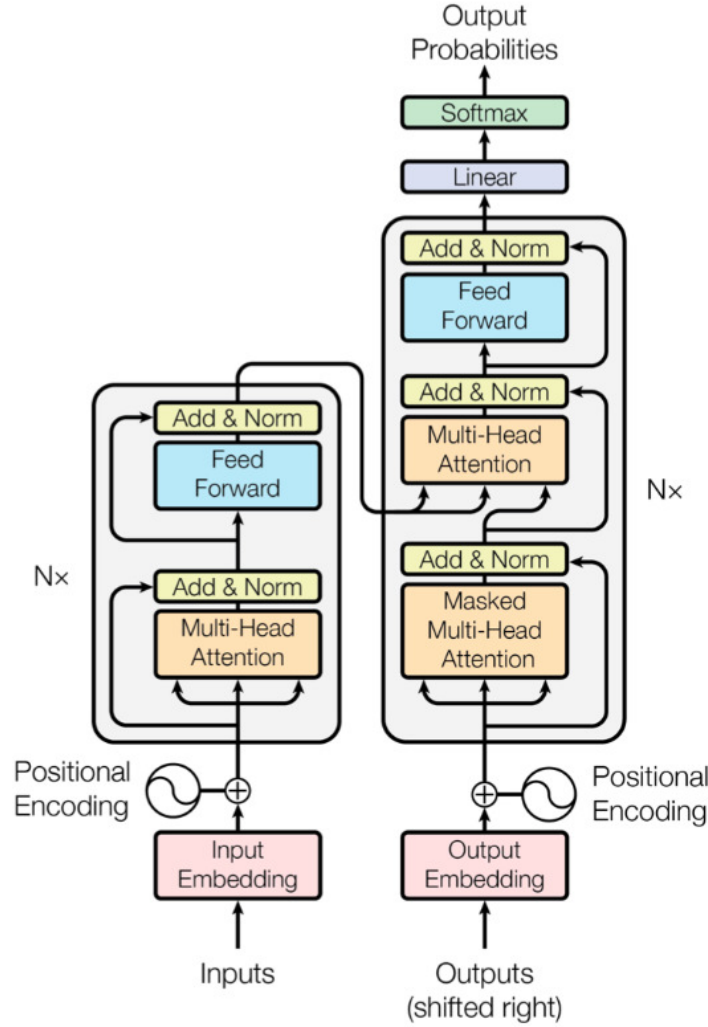


Figure 2.5: Transformer - model architecture [6].

Encoder of Transformer

The Encoder (the Encoder in a Transformer model) is a crucial component designed to process and understand input sequences by focusing on relationships within the data. It consists of multiple identical layers, each containing a multi-head self-attention mechanism and a feed-forward network. The self-attention mechanism allows the encoder to weigh the importance of each token in a sequence relative to others, enabling it to capture

long-range dependencies without the sequential limitations of RNNs. By processing all tokens simultaneously, the encoder can better understand context across the entire input.

Additionally, positional encodings are added to the token embeddings, allowing the encoder to retain information about the sequence's order, despite its parallel structure. As data moves through each encoder layer, it is progressively transformed into higher-level representations, creating rich embeddings for each token. These context-aware representations are then used for tasks like language translation or, in models like BERT [25], directly applied to tasks such as classification or question-answering.

Encoder and Time-Embedding

Unlike natural language data, where positional encoding suffices to capture word order, processing sequential data with Transformers necessitates a nuanced approach to extract temporal dependencies. This gap is bridged by Time Embedding, which imbues the model with an understanding of chronological order, preventing nonsensical predictions where distant historical prices hold equal sway as recent ones. To address the temporal challenges inherent in Transformers, the Time2Vec methodology is adopted, offering a model-agnostic vector representation of time. This approach encapsulates both periodic and non-periodic patterns while maintaining invariance to time rescaling, ensuring the model's ability to comprehend and utilize temporal features effectively in predicting stock prices.

2.7 Geometric Mean Not NaN

In mathematics, the GM (Geometric Mean) is a mean or average that indicates a central tendency of a finite collection of positive real numbers by using the product of their values. The GM is defined as the n^{th} root of the product of n numbers, i.e., for a collection of numbers $a_1, a_2, \dots, a_n$, the GM is

$$\left(\prod_{i=1}^n a_i \right)^{\frac{1}{n}} = \sqrt[n]{a_1 a_2 \cdots a_n}.$$

While combining multiple stocks, we may create the missing data points (NaN values) due to the difference in the size of each stock, so we have to use a proper approach that handles this issue. In this research, we considered GMNN (Figure 2.6). GMNN ensures

that the transformed data will be in the range $[0, 1]$, which is the must-have attribute of the data. GMNN iterates by row and takes the GM of real numbers.

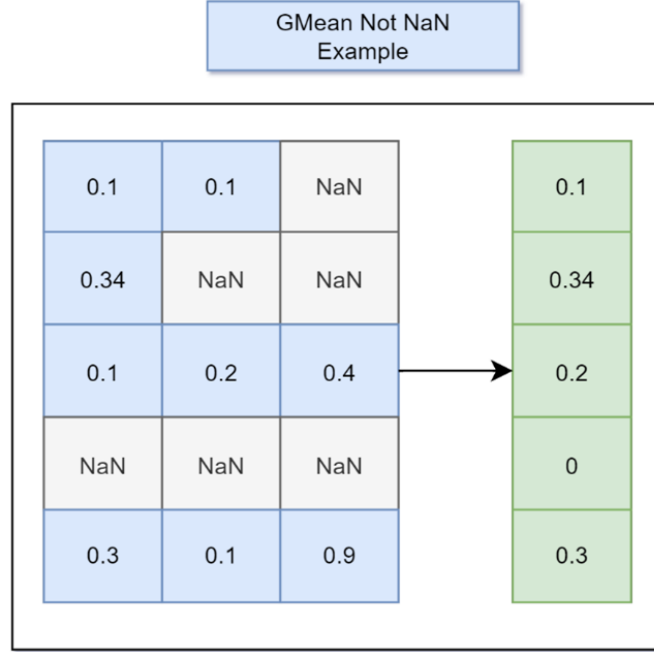


Figure 2.6: Illustration of GMNN.

In our research, GMNN performs column-wise, meaning that it only performs on the related columns (e.g., the “Open” column from table 1 and the “Open” column from table 2), which means with two tables input of size $m \times n$, the result’s size still remains $m \times n$.

2.8 Correlation Analysis

In time series analysis, auto-correlation and cross-correlation are essential tools for examining temporal dependencies within and between time series data. They help identify repeating patterns, trends, and the relationship between different variables over time.

Pearson Correlation

Pearson correlation is a specific type of correlation that measures the linear relationship between two variables. It is closely related to both auto-correlation and cross-correlation when comparing the strength and direction of the relationship between variables. In time series, Pearson correlation is used to quantify how closely two series move together at specific time points or lags. The Pearson correlation coefficient r between two variables x and y is calculated as:

$$r = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2 \sum_{i=1}^n (y_i - \bar{y})^2}},$$

where $\bar{x}$ and $\bar{y}$ are the means of x and y , respectively. Pearson's correlation coefficient r ranges from -1 to 1 ; $r = 1$ indicates a perfect positive linear relationship, $r = -1$ indicates a perfect negative linear relationship, and $r = 0$ suggests no linear relationship.

Auto-correlation

Auto-correlation measures the correlation of a time series with its own lagged values. It is used to detect periodicity or trends within a series. The auto-correlation at lag k is

$$\rho_{xx}(k) = \frac{\sum_{t=1}^{n-k} (x_t - \bar{x})(x_{t+k} - \bar{x})}{\sum_{t=1}^n (x_t - \bar{x})^2},$$

where $\bar{x}$ is the mean of the series. Auto-correlation values range from -1 to 1 , where 1 indicates perfect positive correlation, -1 indicates perfect negative correlation, and 0 indicates no correlation. This measure is valuable for identifying seasonality, trends, or cyclic behavior in a time series.

Cross-correlation

Cross-correlation measures the relationship between two different time series. It assesses how one series correlates with another at various lags, helping to identify lead-lag relationships. The cross-correlation at lag k between two time series x and y is given by:

$$\rho_{xy}(k) = \frac{\sum_{t=1}^{n-k} (x_t - \bar{x})(y_{t+k} - \bar{y})}{\sqrt{\sum_{t=1}^n (x_t - \bar{x})^2 \sum_{t=1}^n (y_t - \bar{y})^2}},$$

where $\bar{x}$ and $\bar{y}$ are the means of the respective series. Cross-correlation is particularly useful in identifying how changes in one time series affect another, for example, in economic or physical systems.

Chapter 3

TT2VFin: Transformer Time2Vec Financial

In this chapter, we introduce a novel deep learning model for financial prediction.

3.1 Motivation

The forecasting finance area is currently under intense research focus: many researchers and specialists are trying to combine Time2Vec with CNNs, RNNs, LSTMs, and Attention mechanism. Moreover, not only finance but also other fields [27, 28].

They usually use only one feature to predict things (for instance, only one stock price to predict its prices). However, using techniques that rely solely on a single feature can be quite challenging when it comes to predicting unexpected financial events, which are often influenced by external factors beyond our control. On the flip side, in the world of finance, it's common practice to focus on identifying and analyzing correlations between specific features and target prices when making investment decisions.

Building upon previous challenges, we were motivated to explore the integration of correlations from multiple features into the Transformer model. We represent a new approach over existing methods by developing a neural network architecture that consists of Time2Vec, residual, multiple attention layers, pooling, and a dense, fully connected part. Given the limitations inherent in financial data, often sparse and messy, we sought to address these issues by selecting multiple stocks exhibiting similar behavior and interconnectedness. The subsequent sections of this paper will detail our approach to time series modeling.

3.2 Methodology

In statistics, correlation is a measure that describes the strength and direction of a relationship between two variables. Correlation coefficients range from -1 to 1 and quantify whether, and how strongly, pairs of variables are related. A positive correlation indicates that as one variable increases, the other tends to increase as well, while a negative correlation indicates an inverse relationship. When two variables are uncorrelated, changes in one variable do not systematically affect the other. Correlations are widely used in fields such as economics, finance, psychology, and biology, where understanding relationships between variables is key for analysis and prediction. The performance of a stock market predictor heavily depends on the correlations between historical data for training and the current input for prediction. The results show below here in Figure 3.1 we use NASDAQ as a base market, Exxon Mobil for Figure 3.2, and Morgan Stanley for Figure 3.3.

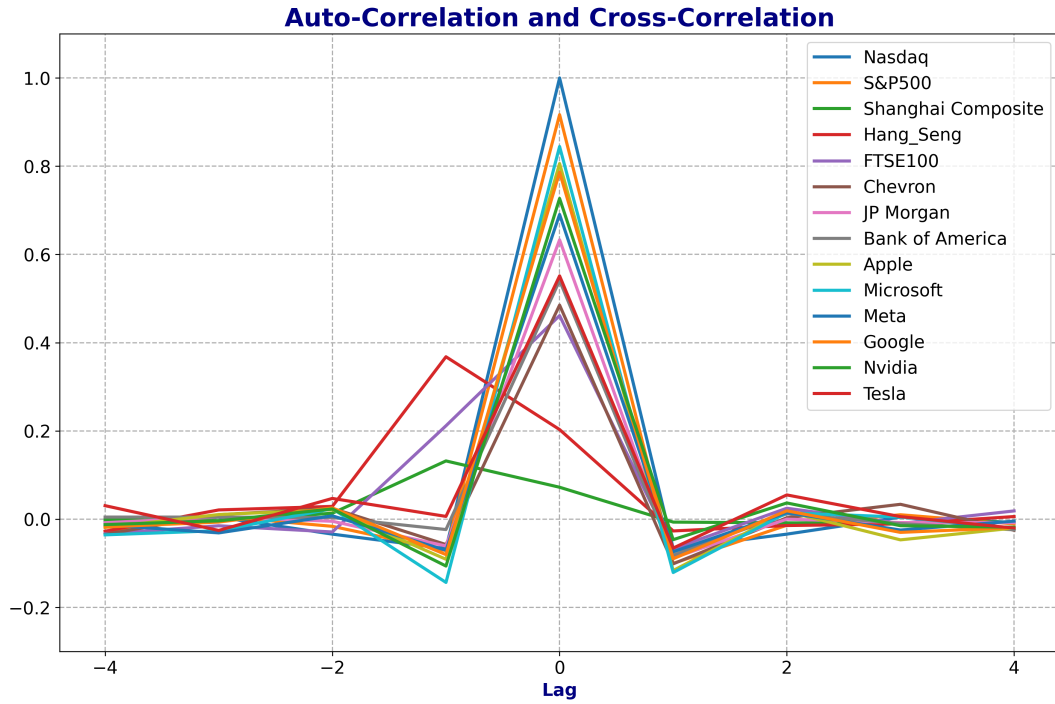


Figure 3.1: Auto-correlation and cross-correlation of market trends using NASDAQ as a base market.

Figure 3.1 reveals that the base market’s auto-correlation is solely non-zero at the origin. This observation suggests that the daily trend of NASDAQ stock approximates a Markov process [29]. Consequently, historical data offers limited insight into its future movements. However, data sources such as S&P500 are promising features for our approach.

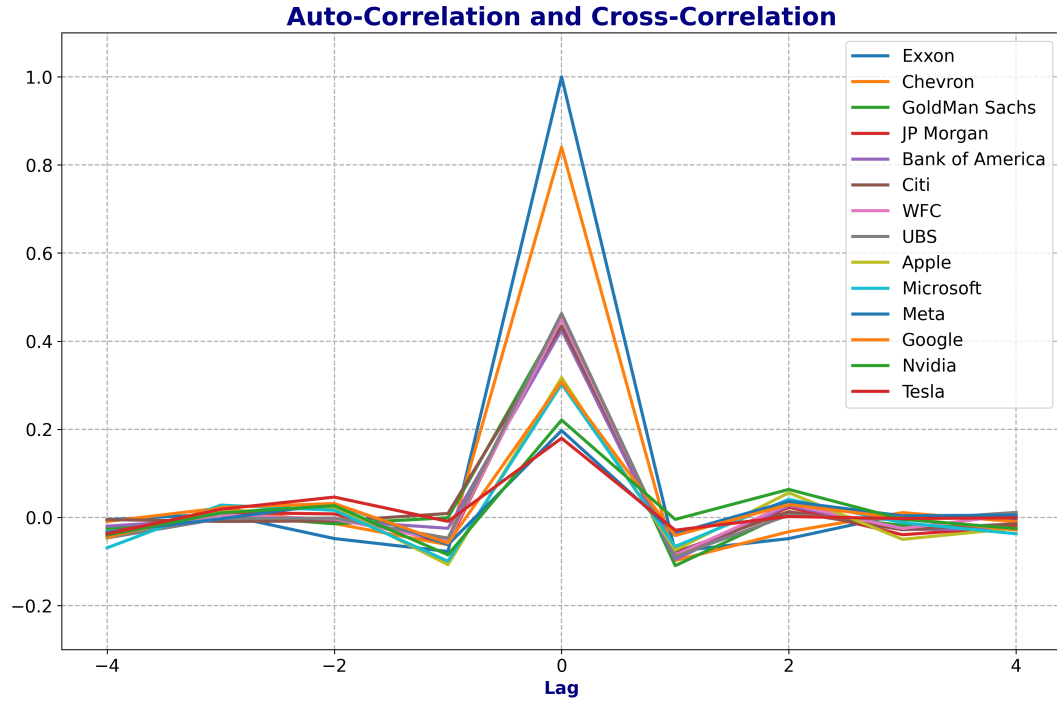


Figure 3.2: Auto-correlation and cross-correlation of market trends using Exxon Mobil as a base market.

From Figure 3.2, Exxon Mobil and Chervon will be a good pair; the same conclusion goes in Figure 3.3; Goldman Sachs will be a good partner for Morgan Stanley.

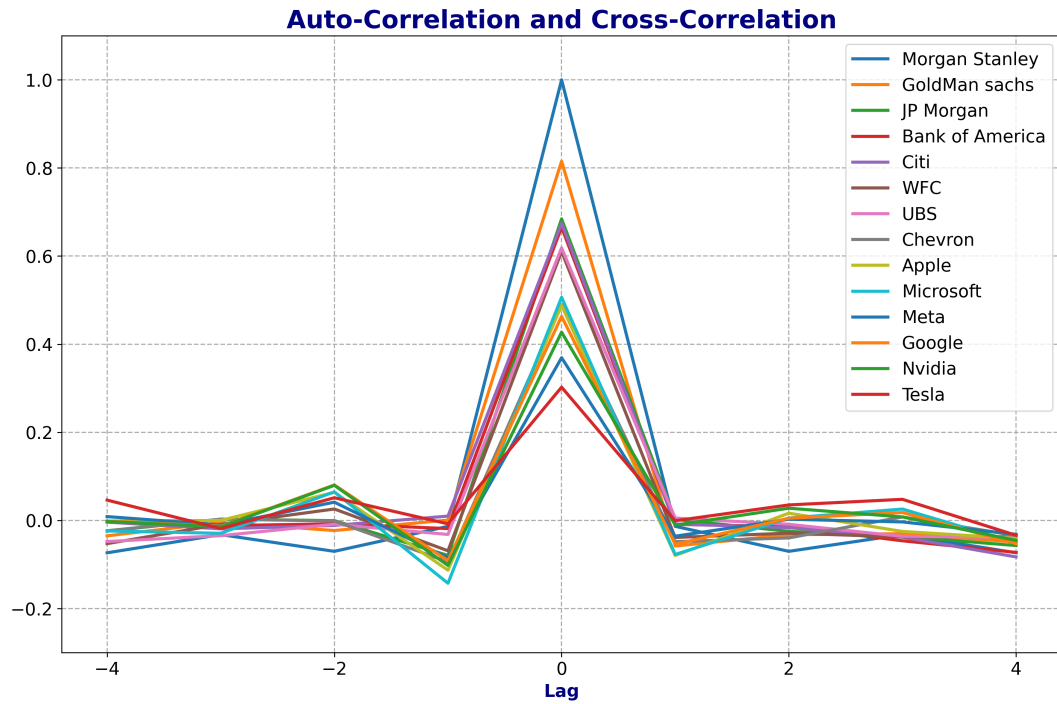


Figure 3.3: Auto-correlation and cross-correlation of market trends using Morgan Stanley as a base market.

Motivated by the strong correlation observed between the selected stock pairs, we propose a novel method to combine multiple stocks into a single input representation. By leveraging cross-correlation relationships, our approach aims to capture the joint dynamics of these correlated stocks, enabling the model to better understand their collective behavior and improve predictive accuracy.

3.3 Data

Data Collection

We downloaded daily stock datasets from their release date up to the 30th of September, 2024, using Yahoo Finance [30]. Each dataset consists of the opening, closing, highest, and lowest stock prices, along with the date and the volume on that specific date. Based on the previous section’s correlation analysis, we collect three groupings of two equities whose members significantly correlated to each other.

Group A: NASDAQ Composite (^IXIC), **S&P500** (^GSPC).

Group B: Exxon Mobil Corporation (XOM), **Chevron Corporation** (CVX).

Group C: Morgan Stanley (MS), **The Goldman Sachs Group, Inc.** (GS).

Data will be split into three sets following a commonly used splitting ratio of 80%:10%:10%. The sets are split in the time order. The training set will be the first 80% of the data. The validation set will be the next 10%. The test set will be the last 10%. Table 3.1 shows an example of downloaded data (NASDAQ).

Date	Open	High	Low	Close	Volume
1971-02-05	100.000000	100.000000	100.000000	100.000000	0
1971-02-08	100.839996	100.839996	100.839996	100.839996	0
1971-02-09	100.760002	100.760002	100.760002	100.760002	0
1971-02-10	100.690002	100.690002	100.690002	100.690002	0
1971-02-11	101.449997	101.449997	101.449997	101.449997	0
⋮	⋮	⋮	⋮	⋮	⋮
2024-09-23	17994.9101	18021.5800	17936.3691	17974.2695	5.237×10^9
2024-09-24	18046.4394	18091.9609	17863.3808	18074.5195	5.347×10^9
2024-09-25	18050.3593	18155.0390	18040.6191	18082.2109	4.858×10^9
2024-09-26	18327.3398	18327.3398	18071.7402	18190.2890	5.669×10^9
2024-09-27	18228.7792	18238.2792	18069.1796	18119.5898	5.367×10^9

Table 3.1: Download data from Yahoo Finance.

Date: The specific day when the trading occurred.

Open: The price of the stock index at the beginning of the trading day (USD).

High: The highest price reached by the stock index during the trading day (USD).

Low: The lowest price reached by the stock index during the trading day (USD).

Close: The price of the stock index at the end of the trading day (USD).

Volume: The total number of shares traded during the day (share).

Data Preprocessing

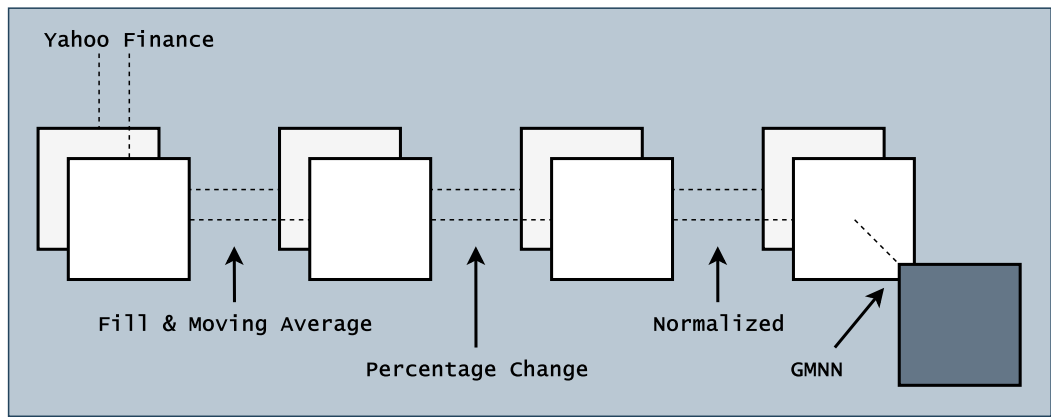


Figure 3.4: Preprocessing data pipeline.

We used moving averages and percentage changes that worked on continuous dates, so discontinuities would cause inconsistency in the processed dataset. To address the obstacle, we first filled forward all missing dates in the dataset. To smooth out the ups and downs in prices, we started by averaging the values over a **14-day** period (Table 3.2).

Table 3.2: NASDAQ data after filling missing dates and applying Moving Average.

Date	Open	High	Low	Close	Volume
1971-02-18	101.385000	101.385000	101.385000	101.385000	6.286×10^7
1971-02-19	101.435000	101.435000	101.435000	101.435000	6.286×10^7
1971-02-20	101.352143	101.352143	101.352143	101.352143	6.286×10^7
1971-02-21	101.269286	101.269286	101.269286	101.269286	6.286×10^7
1971-02-22	101.186429	101.186429	101.186429	101.186429	6.286×10^7
⋮	⋮	⋮	⋮	⋮	⋮
2024-09-23	17646.9049	17745.7008	17542.1035	17681.2809	5.473×10^9
2024-09-24	17725.2477	17821.1216	17617.9785	17756.1837	5.501×10^9
2024-09-25	17795.8869	17873.6079	17707.4627	17805.2324	5.413×10^9
2024-09-26	17861.1333	17925.1522	17759.8498	17849.5616	5.462×10^9
2024-09-27	17907.8133	17962.2335	17795.8906	17880.6766	5.502×10^9

Then, we calculate the percentage change for the next day (Table 3.3).

Table 3.3: NASDAQ data after computing percentage change.

Date	Open	High	Low	Close	Volume
1971-02-19	0.0493%	0.0493%	0.0493%	0.0493%	0%
1971-02-20	−0.0817%	−0.0817%	−0.0817%	−0.0818%	0%
1971-02-21	−0.0818%	−0.0818%	−0.0818%	−0.0818%	0%
1971-02-22	−0.0818%	−0.0818%	−0.0818%	−0.0818%	0%
1971-02-23	−0.0734%	−0.0734%	−0.0734%	−0.0734%	0%
⋮	⋮	⋮	⋮	⋮	⋮
2024-09-23	0.4714%	0.4440%	0.4925%	0.4421%	0.1374%
2024-09-24	0.4439%	0.4250%	0.4325%	0.4236%	0.5009%
2024-09-25	0.3985%	0.2945%	0.5079%	0.2762%	−1.5955%
2024-09-26	0.3666%	0.2884%	0.2958%	0.2490%	0.9052%
2024-09-27	0.2613%	0.2069%	0.2029%	0.1743%	0.7388%

When building our model, we make sure to scale these percentage changes so they fall between 0 and 1; using a standard min-max scaling method, we get the result (Table 3.4).

$$\mathbf{x}_{\text{standarlized}} = \frac{\mathbf{x} - \min(\mathbf{x})}{\max(\mathbf{x}) - \min(\mathbf{x})} \quad (3.1)$$

Table 3.4: NASDAQ data after normalizing.

Date	Open	High	Low	Close	Volume
1971-02-19	0.635564	0.635564	0.635564	0.635564	0.481983
1971-02-20	0.606946	0.606946	0.606946	0.606946	0.481983
1971-02-21	0.606932	0.606932	0.606932	0.606932	0.481983
1971-02-22	0.606917	0.606917	0.606917	0.606917	0.481983
1971-02-23	0.608753	0.608753	0.608753	0.608753	0.481983
⋮	⋮	⋮	⋮	⋮	⋮
2024-09-23	0.727778	0.721795	0.732374	0.721381	0.489125
2024-09-24	0.721774	0.717637	0.719280	0.717335	0.508016
2024-09-25	0.711851	0.689130	0.735748	0.685136	0.399052
2024-09-26	0.704885	0.687790	0.689421	0.679179	0.529034
2024-09-27	0.681884	0.669982	0.669123	0.662872	0.520383

Repeat the steps with the other stock index (S&P500 in this case), then we push all the processed data (Table 3.4) to the GMNN block (Figure 2.6), which will be the final input to feed for the model (Table 3.5).

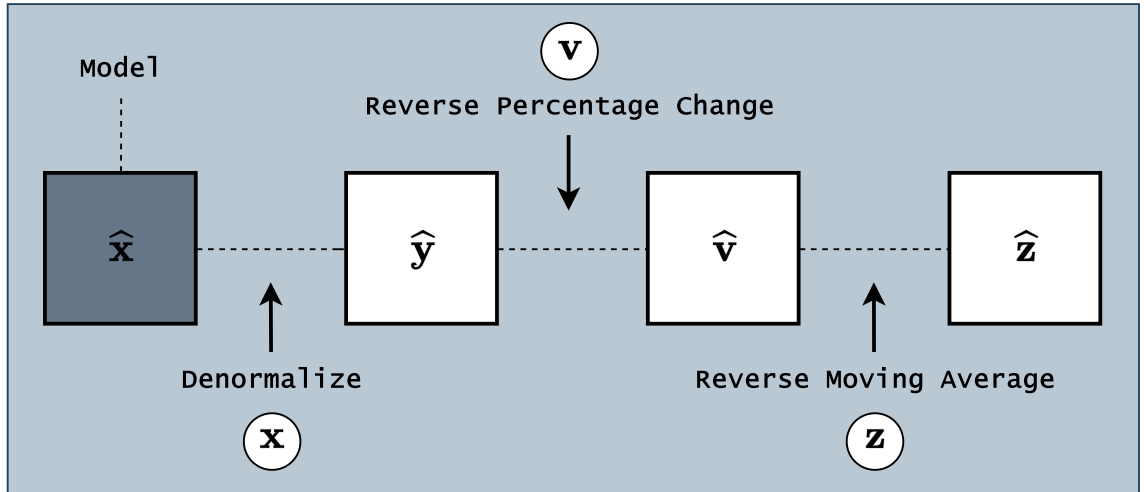
Table 3.5: Combine NASDAQ and S&P500 using GMNN block.

Date	Open	High	Low	Close	Volume
1928-01-13	0.521913	0.521913	0.521913	0.521913	0.458965
1928-01-14	0.490008	0.490008	0.490008	0.490008	0.458965
1928-01-15	0.489934	0.489934	0.489934	0.489934	0.458965
1928-01-16	0.489860	0.489860	0.489860	0.489860	0.458965
1928-01-17	0.490609	0.490609	0.490609	0.490609	0.458965
⋮	⋮	⋮	⋮	⋮	⋮
2024-09-23	0.659504	0.652571	0.661733	0.653370	0.470005
2024-09-24	0.652042	0.650059	0.653904	0.650004	0.483189
2024-09-25	0.647456	0.629080	0.667998	0.625579	0.425468
2024-09-26	0.639651	0.626475	0.630009	0.620237	0.502521
2024-09-27	0.621775	0.612884	0.612355	0.607554	0.493897

Data Postprocessing

Data Postprocessing is only used for predicting trends and stock prices in section 4.2; we do not use this for hyperparameters tuning (Table 3.7) and models evaluation (chapter 4).

Figure 3.5: Postprocessing pipeline.



The output from the model $\hat{\mathbf{x}}$ may contain values that are not in the range $[0,1]$. To address this situation, we will clip $\hat{\mathbf{x}}$ to fall in the range $[0,1]$. Figure 3.5 describes the postprocessing pipeline. Let's walk through each step.

1. Denormalization

$$\hat{\mathbf{y}}_i = \begin{cases} \mathbf{x}_i \times (M - m) + m & \text{if } 0 \leq i < 128 \\ \Pi_{[0,1]}(\hat{\mathbf{x}}_{i-128}) \times (M - m) + m & \text{if } i \geq 128 \end{cases} \quad (3.2)$$

where $\Pi_{[0,1]}$ is the clip function, $\mathbf{x}$ is the original normalized values, $\hat{\mathbf{x}}$ is the predicted normalized values (output from model), $\hat{\mathbf{y}}$ is the predicted percentage change values, and M is the max, m is the min (from Equation 3.1).

2. Reverse percentage change:

$$\hat{\mathbf{v}}_i = \begin{cases} \mathbf{v}_i & \text{if } i = 0 \\ \mathbf{v}_{i-1} \times (1 + \hat{\mathbf{y}}_{i-1}) & \text{if } i \geq 1 \end{cases} \quad (3.3)$$

where $\mathbf{v}$ is the original moving average values, $\hat{\mathbf{v}}$ is the predicted moving average values, and $\hat{\mathbf{y}}$ is the predicted percentage change values (from Equation 3.2).

3. Reverse moving average:

$$\hat{\mathbf{z}}_i = \begin{cases} \mathbf{z}_i & \text{if } 0 \leq i < 13 \\ 14 \times \hat{\mathbf{v}}_i - \sum_{j=i-13}^{i-1} \mathbf{z}_j & \text{if } i \geq 13 \end{cases} \quad (3.4)$$

where $\mathbf{z}$ is the original close prices, $\hat{\mathbf{z}}$ is the predicted close prices, and $\hat{\mathbf{v}}$ is the predicted moving average values (from Equation 3.3).

3.4 Proposed Model

Inspired by the inherent limitations of complex deep learning models, which are frequently trained on historically inconsistent and unreliable datasets, as well as the common data challenges across diverse industries such as technology, finance, energy, and capital markets, this research aims to tackle critical methodological constraints in predictive analytics.

We introduce a novel network architecture, presented in Figure 3.6, that combines the Time2Vec and the Encoder.

1. **Input:** Take the processed data of shape (batch_size, seq_len, 5). Where 5 is the number of features in the data (Open, High, Low, Close, Volume).
2. **Time2Vec:** This layer contains 4 sub-layers.
 - (a) **Linear:** Capturing linear trends in prices overtime – which are not periodic but still play an important role in time series.

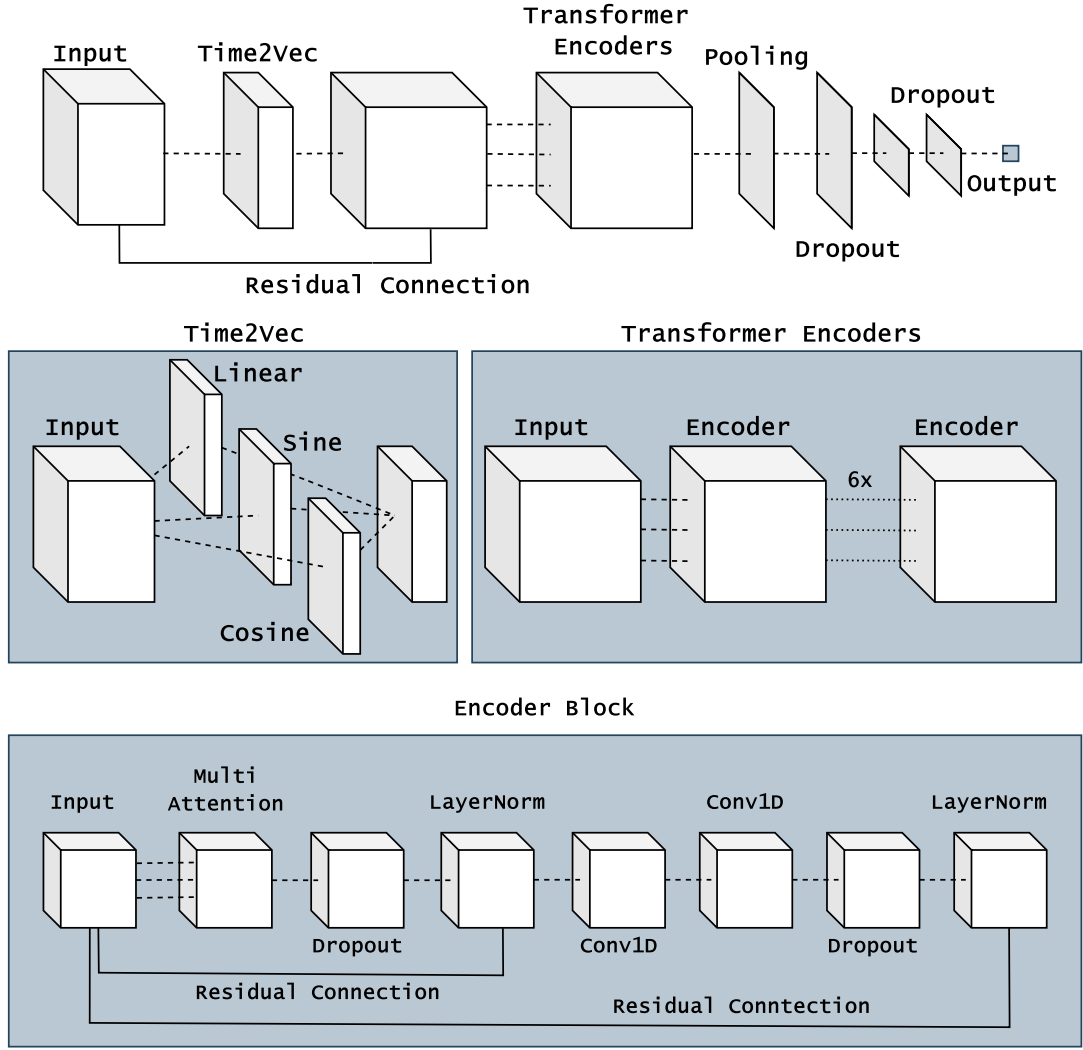


Figure 3.6: Proposed model.

(b) **Sine** and **Cosine**:

- These layers are important for our model. Their task is encoding positions, which we can consider as time – representing continuous time instead of discrete positions. They also capture the periodic behaviors of prices.
- In the Time2Vec paper, Kazemi et al. use only sine as a periodic function. By experimenting, we realize that using cosine as an additional periodic function will yield a better result in this study.

(c) **Concat**: Concatenating outputs from the above three layers.

3. **Concat**: Performs a residual connection [31] between the input with the output from Time2Vec. This action significantly improves the behavior of the network, which makes the Time2Vec layer more valuable and powerful to our architecture.

4. **Transformer Encoders:** This layer contains 6 Encoder blocks inside.
5. **Encoder:** Encoder contains:
 - (a) **MultiAttention:** Contains h SingleAttention blocks.
 - (b) **LayerNorm:** We follow the PostLN that was described by Vaswani et al.
 - (c) **Conv1D:** This is a positional-wise feed-forward layer. It helps the model focus on local dependencies in time series data. Conv1D is more parameter-efficient and can capture local trends in sequences, which are often important in stock price prediction.
 - (d) **Dropout:** Prevent overfitting the model.
6. **Pooling:** Reducing the spatial dimensions of the output from the Encoder, in terms of the number of features, while retaining the most important information.
7. **Dropout:** Prevent overfitting the model.
8. **Dense:** Dense is a fully connected neural network layer. It is used here to decrease the output dimension.

In *Attention Is All You Need* [6], Vaswani et al. suggests people to follow the formula for hyperparameters where

$$d_{model} = d_k \times h,$$

This ensures that the dimension of the multi-head attention layer's output equals d_{model} , which simplifies residual connections and allows for consistent dimension throughout the model. Here, we suggest the hyperparameters of Table 3.6, in accordance with the hyperparameter tuning in Section 4.3, and based on the following key differences to the general Transformer architecture:

Table 3.6: Suggested values for hyperparameters.

Hyperparameter	Suggested value	Description
B	32	Batch size
S	128	Sequence length
N	50	Number of epochs
α	10^{-3}	Learning rate
p	0.2	Dropout rate
$d_k/d_q/d_v$	64	Key/Query/Value's dimension
h	8	Number of head in the multi-head attention
d_{ff}	256	Feed-forward layer's dimension
d_{model}	8	Encoder input's dimension

- The original Transformer architecture was designed for tasks like translating, where capturing long-range dependencies and complex contextual information is key. It relies on its ability to model fine-grained relationships between sequences with a high-dimensional attention mechanism, often using large d_{model} sizes (e.g., 512 or 1024).
- In the context of financial prediction, where the input data consists of historical financial data, the model does not require as large a model size. Our choice of $d_{model} = 5$ (number of features) + 3 (Time2Vec output dimension) = 8 describes a more compact model that is beneficial for avoiding overfitting, especially when our dataset is not large.

To resolve the problem, at the end of each multi-head attention layer, we apply a linear transformation to reduce the dimension from $d_k \times h = 512$ down to 8 so that the residual connection can work smoothly.

We change the second residual connection, compare to the original one from the paper.

- **Enhanced Stability with Input Persistence:** By consistently adding the input after each transformation, we ensure the model retains essential base information about the stock's behaviors. Stock price tends to have strong temporal dependencies, and unaltered input data may help the model avoid losing key information about past prices that attention mechanisms dilute and distort.
- **Regularization-like Effect:** This method is viewed as a regularizer, restricting the model's propensity to overfit to attention's noise. By balancing reliance on modified features and steady input, the model generalizes well and avoids pursuing noisy patterns, which can lead to overfitting.

3.5 Hyperparameters Tuning

We can suggest a set of hyperparameters in Table 3.6 since we chose group B to perform a grid search to find the optimal hyperparameters for our model using the test set. The architecture hyperparameters are d_{model} , d_k , d_v , d_q , h , d_{ff} , and p – where $d_{model} := 8$ is fixed; $d_v = d_q = d_k$ (self-attention) are 32, 64, or 128; h is 4, 8, or 12; d_{ff} is 256, 512, or 1024; and p is 0.1 or 0.2. The training hyperparameters are B , S , N , and α – where $B := 32$, and $S := 128$ are fixed; N is 50 or 75; α is 10^{-3} or 10^{-4} .

Table 3.7: Hyperparameters Tuning.

N	p	α	d_k	h	d_{ff}	P	Exxon Mobil		Chervon	
							RMSE	MAPE	RMSE	MAPE
50	0.1	10^{-3}	32	4	256	62K	0.0869	11.292	0.0702	10.045
75	0.1	10^{-3}	32	4	512	88K	0.0799	6.0019	0.0648	5.1607
75	0.1	10^{-3}	32	4	1024	141K	0.0682	8.4010	0.0635	9.9112
50	0.1	10^{-3}	32	8	256	89K	0.0870	11.302	0.0701	10.029
50	0.1	10^{-3}	32	8	512	115K	0.0643	7.9005	0.0403	5.3803
50	0.2	10^{-3}	32	8	1024	167K	0.0646	5.8632	0.0503	4.2050
75	0.2	10^{-4}	32	12	256	116K	0.0654	7.5044	0.043	5.5131
75	0.2	10^{-4}	32	12	512	142K	0.0587	6.7315	0.0476	6.6845
75	0.2	10^{-4}	32	12	1024	194K	0.0378	4.5500	0.0291	3.7463
50	0.1	10^{-3}	64	4	256	89K	0.0868	11.287	0.0703	10.054
75	0.2	10^{-4}	64	4	512	115K	0.0359	4.3287	0.0300	3.7651
75	0.2	10^{-4}	64	4	1024	167K	0.0732	6.3290	0.0576	5.0687
50	0.2	10^{-3}	64	8	256	143K	0.0333	4.0724	0.0275	3.3767
75	0.2	10^{-4}	64	8	512	169K	0.0795	6.2275	0.0677	5.8540
75	0.2	10^{-4}	32	12	1024	221K	0.0357	4.4707	0.0362	4.8245
75	0.2	10^{-4}	64	12	256	197K	0.0334	4.2380	0.0332	4.9536
75	0.2	10^{-4}	64	12	512	223K	0.0364	4.3115	0.0292	3.7155
75	0.2	10^{-4}	64	12	1024	275K	0.0513	6.3385	0.0354	4.8013
75	0.2	10^{-4}	128	4	256	143K	0.0641	7.3585	0.0584	8.8346
75	0.2	10^{-4}	128	4	512	169K	0.0381	4.5523	0.0309	4.1487
75	0.2	10^{-4}	128	4	1024	221K	0.0669	7.5087	0.0508	6.6189
75	0.2	10^{-4}	128	8	256	250K	0.0589	7.4174	0.0511	7.9066
75	0.2	10^{-4}	128	8	512	277K	0.0390	4.5403	0.0316	4.1567
75	0.2	10^{-4}	128	8	512	329K	0.0448	5.5919	0.0357	4.7924
75	0.2	10^{-4}	128	12	256	358K	0.0375	4.9504	0.0421	6.4951
75	0.2	10^{-4}	128	12	512	384K	0.0406	4.8130	0.0312	4.1356
75	0.2	10^{-4}	128	12	1024	436K	0.0382	4.6317	0.0293	3.7778

For the sake of computational efficiency, we followed the heuristic to prioritize architectural hyperparameters (excluding p) over training ones. To do this, we increase N and p while decreasing α when the total number of parameters P is high, and vice versa. This means that training hyperparameters and p can be increased or decreased. The statistics are presented in Table 3.7.

By looking at the statistics, we can find that the set of hyperparameters where $d_{model} := 8$, $d_k := 64$, $h := 8$, $d_{ff} := 256$, $p := 0.2$, $B := 32$, $S := 128$, $N := 50$, $\alpha := 10^{-3}$ provides the best performing model. In addition, the total number of parameters $P := 143K$ is not large at all, which shows that this is the lightweight model too.

Chapter 4

Result and Evaluation

In this section we present and discuss our prediction results. First, we trained and optimized the proposed model using the training and validation sets, then we evaluated our models by using the test set (the last 10% of the data). Models such as NAS, SP, EM, C, MS, and GS are single-feature models, meaning that we only use their target for training; on the other hand, NAS_SP, EM_C, and MS_GS are multi-feature models; they are trained by using their combined dataset (e.g., NASDAQ and S&P500 for NAS_SP). To evaluate models in each group, we rely on their target, for instance: NAS is a single-feature model with NASDAQ as a target, NAS_SP is a multi-feature model with NASDAQ as a target either. Then to know how good NAS_SP is better than NAS, we will compare them with respect to the same target which is NASDAQ. By this evaluation, we expect to figure out how well the cross-correlation (section 3.2) affects the models' performance.

Table 4.1: Evaluations by target of each group.

Group	Target	Model	RMSE	MSE	MAPE	MAE	R2
A	NASDAQ	NAS	0.0291	0.0008	3.1392	0.0187	0.8264
		NAS_SP	0.0248	0.0006	2.6785	0.0161	0.8743
	S&P500	NAS_SP	0.0164	0.0003	1.9422	0.0098	0.8766
		SP	0.0180	0.0003	2.0511	0.0104	0.8517
B	Exxon Mobil	EM	0.0350	0.0012	4.3509	0.0204	0.8163
		EM_C	0.0333	0.0011	4.0724	0.0200	0.8330
	Chervon	EM_C	0.0275	0.0008	3.3767	0.0149	0.8241
		C	0.0342	0.0012	3.7204	0.0172	0.7293
C	Morgan Stanley	MS	0.0148	0.0002	1.7162	0.0105	0.8655
		MS_GS	0.0166	0.0003	2.0747	0.0128	0.8325
	Goldman Sachs	MS_GS	0.0221	0.0005	2.8797	0.0156	0.8545
		GS	0.0201	0.0004	2.6159	0.0144	0.8797

In Table 4.1, RMSE, MSE, MAPE, and MAE are min metrics (i.e., the smaller the better), and R2 is the max metric (i.e., the larger the better). Overall, multi-feature models outperform single-feature models. However, in circumstances where the stock ages are young (i.e., their release date is about 10 to 20 years ago) and the cross-correlations are not tight enough, the single-feature model may outperform the multi-feature model. In addition, by returning to Figure 3.1, Figure 3.2, and Figure 3.3, we can point out that the cross-correlation of NASDAQ and S&P500 is approximately 91%, Exxon Mobil and Chervon is about 84%, and Morgan Stanley and Goldman Sachs is about 81%. Based on these facts, we can conclude that the better the link between stocks, the better the prediction. Choosing equities with a high cross-correlation ($\approx 84\%+$) yields higher quality results.

4.1 Comparing to SOTA models

In the following tables (Table 4.2, Table 4.3, Table 4.4), TT2VFin refers to our basic model that was introduced in earlier chapters. We will compare TT2VFin versions in which TT2VFin+p uses the positional encoding layer, TT2VFin+M uses the mask in the attention mechanism, and TT2VFin+p+M uses both positional encoding and masking. We also compare our model to models utilizing RNN and LSTM architectures from section 2.2.

Table 4.2: Comparing to SOTA models – Group A.

Target	Model	P	RMSE	MSE	MAPE	MAE	R2
NASDAQ	TT2VFin	143K	0.0248	0.0006	2.6785	0.0161	0.8743
	TT2VFin+M	143K	0.0293	0.0009	3.3222	0.0205	0.8240
	TT2VFin+p	143K	0.1773	0.0314	27.500	0.1691	- 5.427
	TT2VFin+p+M	143K	0.0337	0.0011	3.9642	0.0241	0.7685
	RNN	559K	0.0316	0.0010	3.6215	0.0224	0.7959
	LSTM	355K	0.0272	0.0007	2.9289	0.0176	0.8490
S&P500	TT2VFin	143K	0.0164	0.0003	1.9422	0.0098	0.8766
	TT2VFin+M	143K	0.0180	0.0003	2.2002	0.0109	0.8515
	TT2VFin+p	143K	0.2265	0.0513	42.337	0.2253	- 22.51
	TT2VFin+p+M	143K	0.0247	0.0006	3.6617	0.0189	0.7204
	RNN	559K	0.0195	0.0004	2.3061	0.0116	0.8262
	LSTM	355K	0.0184	0.0003	2.3468	0.0119	0.8448

In Table 4.2, we see that the proposed model TT2VFin outperforms its variations. Applying a mask does not significantly affect TT2VFin performance; however, the po-

sitional encoding layer confuses the model and produces poor results. TT2VFin outperforms RNN despite utilizing around three times as many parameters as RNN. LSTM demonstrates its power in financial prediction; nonetheless, there is still a performance difference between TT2VFin and LSTM. TT2VFin outperforms LSTM by around $9.25\% \pm 0.63\%$ (averaging the metrics).

Table 4.3: Comparing to SOTA models – Group B.

Target	Model	P	RMSE	MSE	MAPE	MAE	R2
Exxon Mobil	TT2VFin	143K	0.0333	0.0011	4.0724	0.0200	0.8330
	TT2VFin+M	143K	0.0460	0.0021	5.8378	0.0303	0.6824
	TT2VFin+p	143K	0.0859	0.0074	11.169	0.0619	-0.108
	TT2VFin+p+M	143K	0.0870	0.0076	11.315	0.0632	-0.137
	RNN	559K	0.0351	0.0012	4.5526	0.0239	0.815
	LSTM	355K	0.0342	0.0012	4.3279	0.0224	0.8248
Chervon	TT2VFin	143K	0.0275	0.0008	3.3767	0.0149	0.8241
	TT2VFin+M	143K	0.0380	0.0014	5.2774	0.0237	0.6661
	TT2VFin+p	143K	0.0713	0.0051	10.285	0.0500	-0.176
	TT2VFin+p+M	143K	0.0700	0.0049	10.007	0.0486	-0.134
	RNN	559K	0.0287	0.0008	3.6525	0.0172	0.8093
	LSTM	355K	0.0274	0.0008	3.4605	0.0158	0.8261

In Table 4.3, we get the same result as Table 4.2, where TT2VFin is dominating over others. TT2VFin+p and TT2VFin+M have no change in performance, but TT2VFin+p+M cannot maintain its previous performance. We can therefore conclude that conventional positional encoding is inappropriate for our architecture. RNN and LSTM remain in their positions, despite LSTM producing slightly better outcomes than previously.

Table 4.4: Comparing to SOTA models – Group C.

Target	Model	P	RMSE	MSE	MAPE	MAE	R2
Morgan Stanley	TT2VFin	143K	0.0166	0.0003	2.0747	0.0128	0.8325
	TT2VFin+M	143K	0.0174	0.0003	2.1018	0.0130	0.8152
	TT2VFin+p	143K	0.0495	0.0024	6.4577	0.0409	-0.494
	TT2VFin+p+M	143K	0.0484	0.0023	6.3163	0.0399	-0.430
	RNN	559K	0.0258	0.0007	3.6253	0.0227	0.5933
	LSTM	355K	0.0186	0.0003	2.4826	0.0149	0.7891
Goldman Sachs	TT2VFin	143K	0.0221	0.0005	2.8797	0.0156	0.8545
	TT2VFin+M	143K	0.0236	0.0006	3.2474	0.0176	0.8343
	TT2VFin+p	143K	0.0661	0.0044	10.086	0.0521	-0.304
	TT2VFin+p+M	143K	0.0671	0.0045	10.265	0.0529	-0.343
	RNN	559K	0.0246	0.0006	3.4026	0.0192	0.8200
	LSTM	355K	0.0280	0.0008	4.3016	0.0229	0.7669

In Table 4.4, our conclusion remains true: TT2VFin is the best among all, followed by TT2VFin+M. TT2VFin+p and TT2VFin+p+M are the worst. RNN continues to perform well, but it cannot yet compete with our model. LSTM, on the other hand, does not produce outstanding results despite having more than twice the number of parameters as our base model.

In conclusion, our base model – TT2VFin – uses 2 to 3 times fewer parameters than others but has roughly 14.23% greater performance (by averaging all the metrics differences between base and RNN and LSTM). Applying positional encoding confuses the TT2VFin rather than improving it. Applying the mask does not significantly degrade performance, but it also does not increase it.

4.2 Predicting Trends and Prices

We want to achieve the results that are the predicted trends and prices of the stocks. By leveraging the postprocessing that is presented in section 3.3 and using TT2VFin models, we can achieve the expected values.

To predict stock trends, Moving Average prices' differences from the test sets are transformed into classes; this is a classification task where the categories are **Up** Trend, **Stable** Trend, and **Down** Trend original, but **Stable** seems almost never happening, so we decided to use only **Up** and **Down** as two classes for this task. The proposed model achieved approximately **87-90%** classification accuracy of the trend prediction task. We also provide the confusion matrices for detailed evaluation (see Figure 4.1, Figure 4.3, Figure 4.5, Figure 4.7, Figure 4.9, and Figure 4.11).

For the predicting stock's close prices task, we use the 6-month date range from the the 1st of May to the the 1st of November, 2024 to challenge the models (given that they were trained on the data up to the 30th of September, 2024). We also display moving average values, a smoothed-out version of the prices; these lines can be considered as a “trend line” over time (see Figure 4.2, Figure 4.4, Figure 4.6, Figure 4.8, Figure 4.10, and Figure 4.12).

NASDAQ

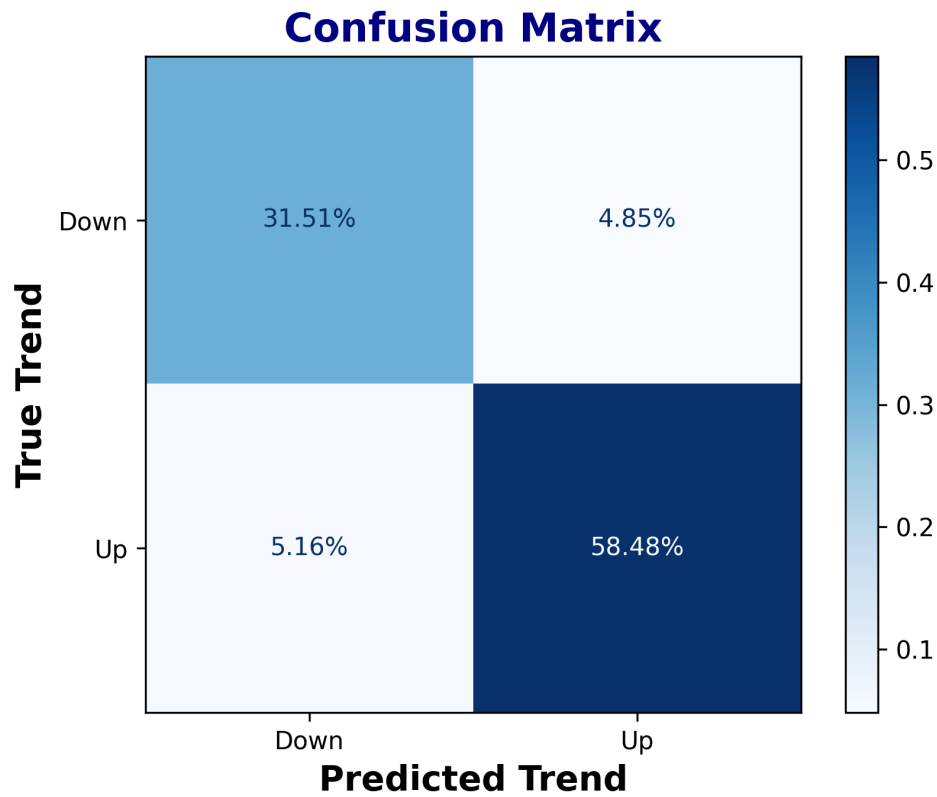


Figure 4.1: Trend confusion matrix of NASDAQ with total accuracy of 89.99%.

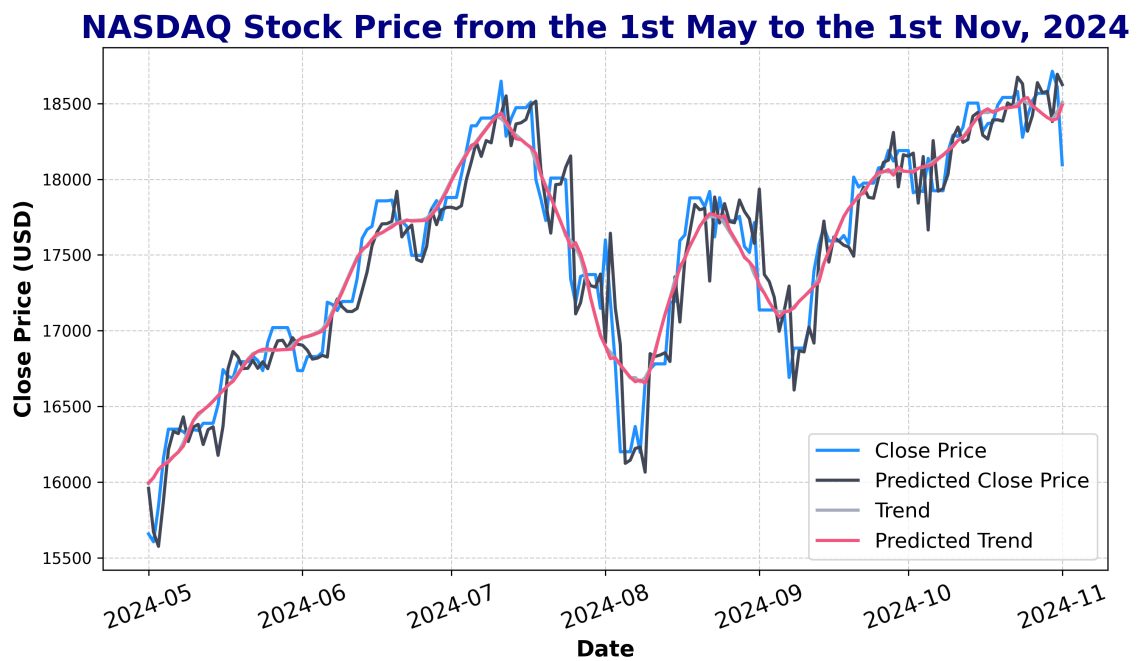


Figure 4.2: NASDAQ Stock Price from the 1st May to the 1st Nov, 2024.

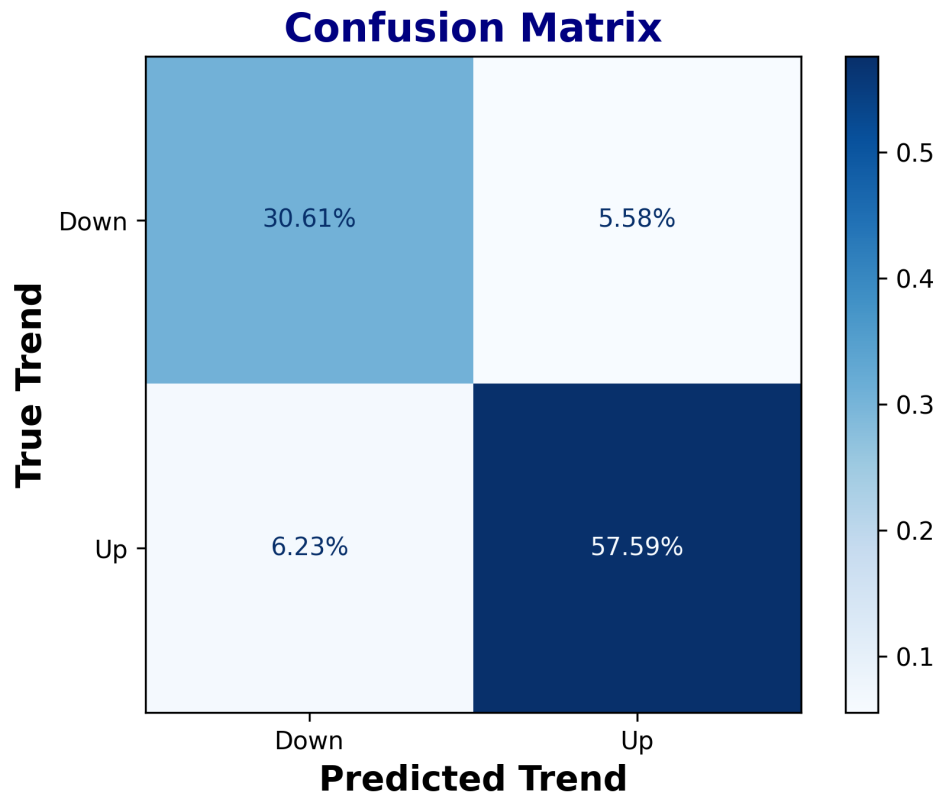
S&P500

Figure 4.3: Trend confusion matrix of S&P500 with total accuracy of 88.20%.

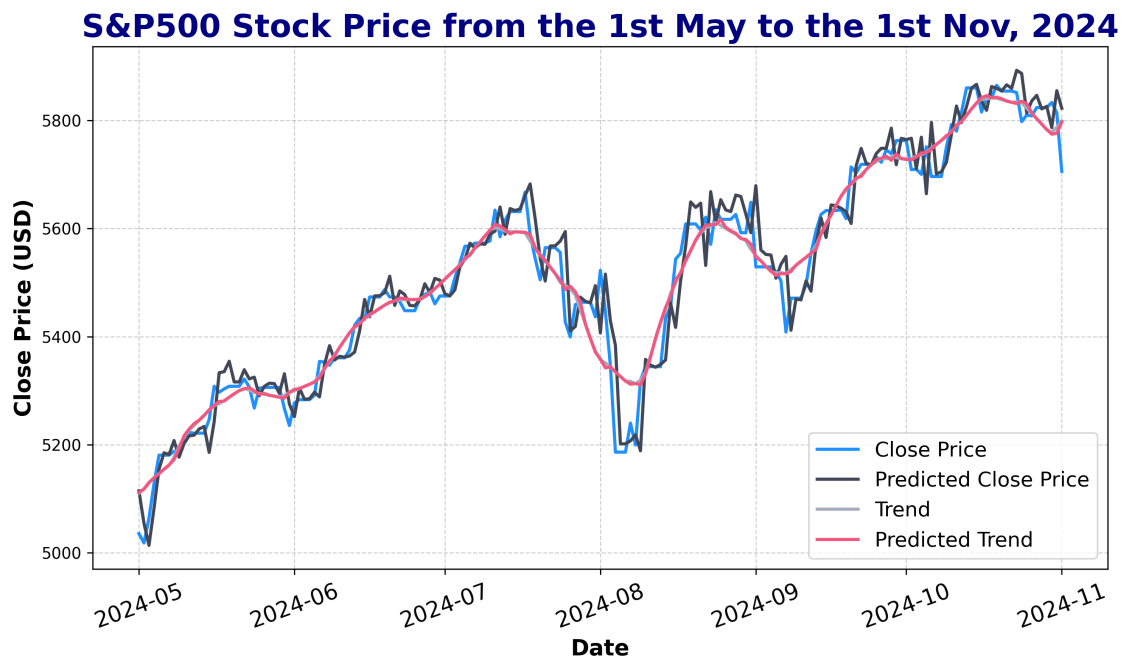


Figure 4.4: S&P500 Stock Price from the 1st May to the 1st Nov, 2024.

Exxon Mobil

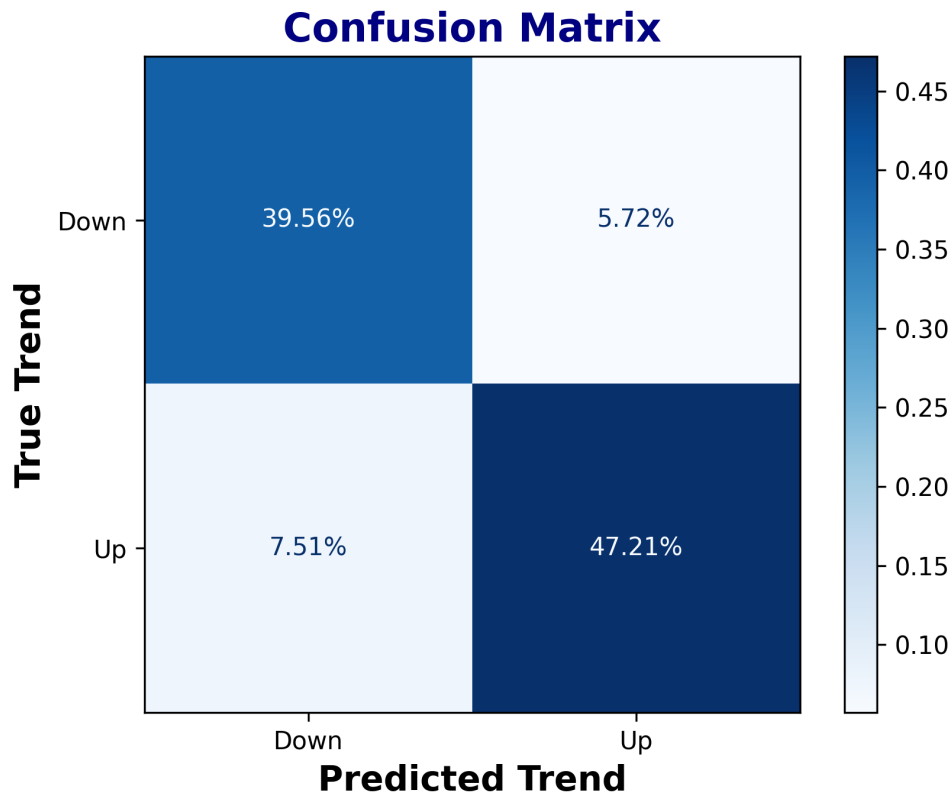


Figure 4.5: Trend confusion matrix of Exxon Mobil with total accuracy of 86.77%.

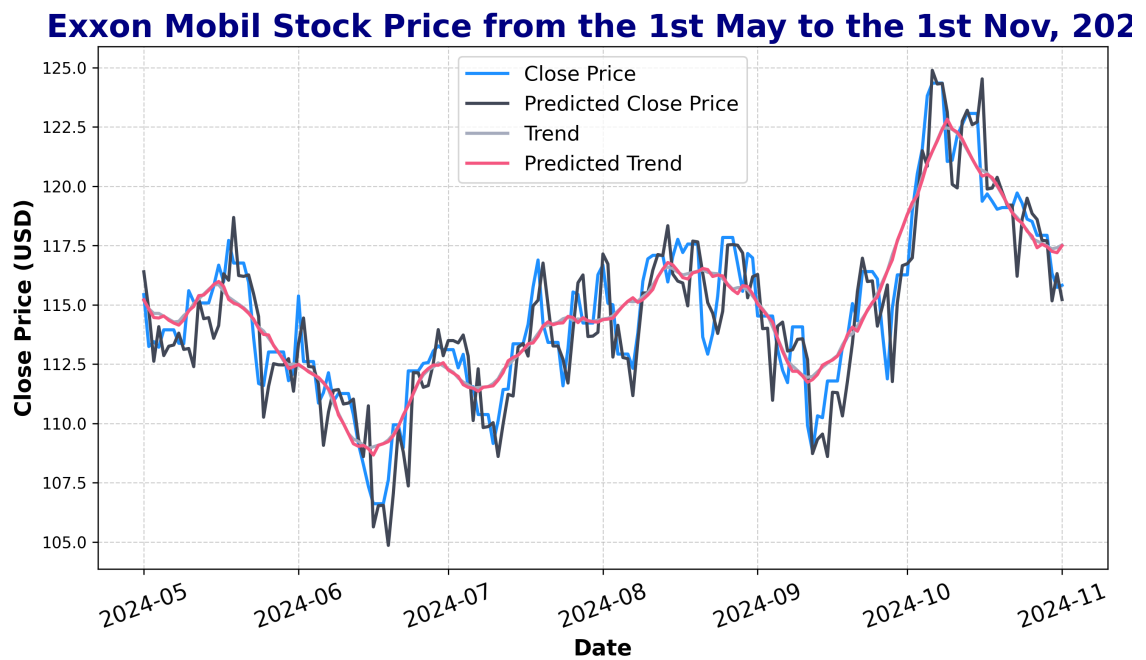


Figure 4.6: Exxon Mobil Stock Price from the 1st May to the 1st Nov, 2024.

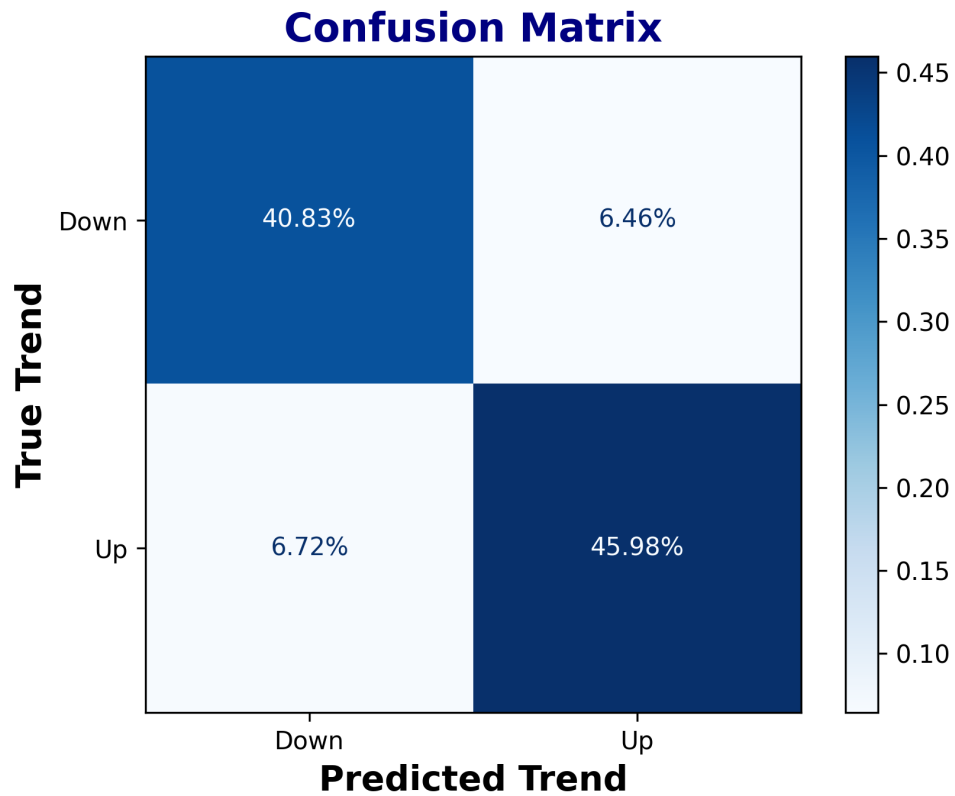
Chervon

Figure 4.7: Trend confusion matrix of Chervon with total accuracy of 86.81%.

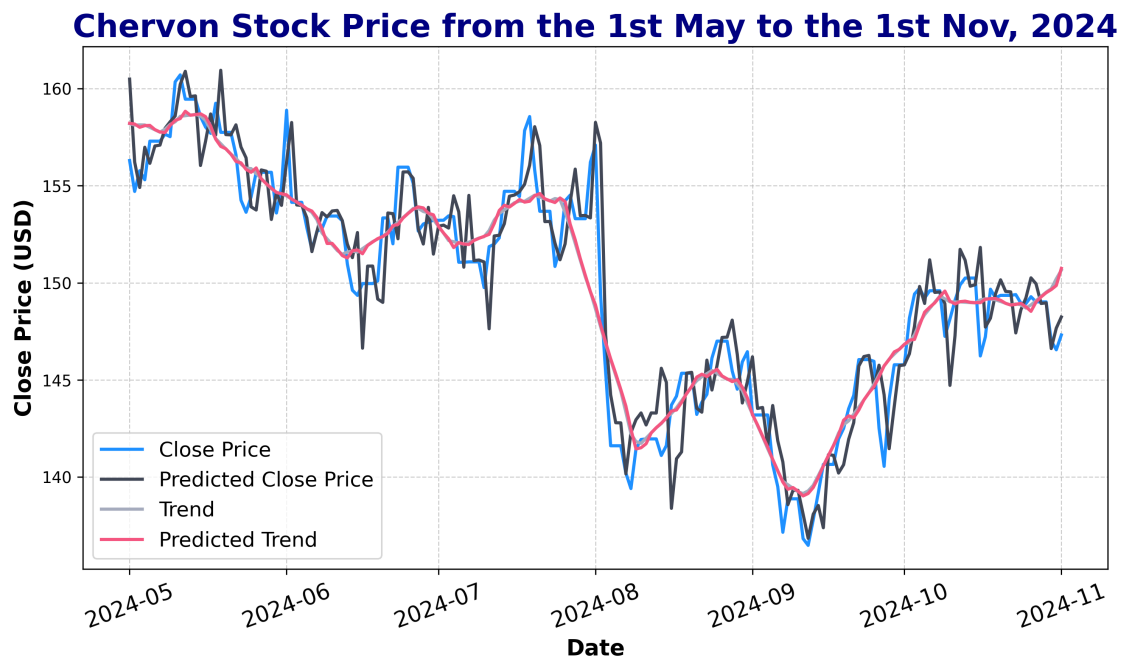


Figure 4.8: Chervon Stock Price from the 1st May to the 1st Nov, 2024.

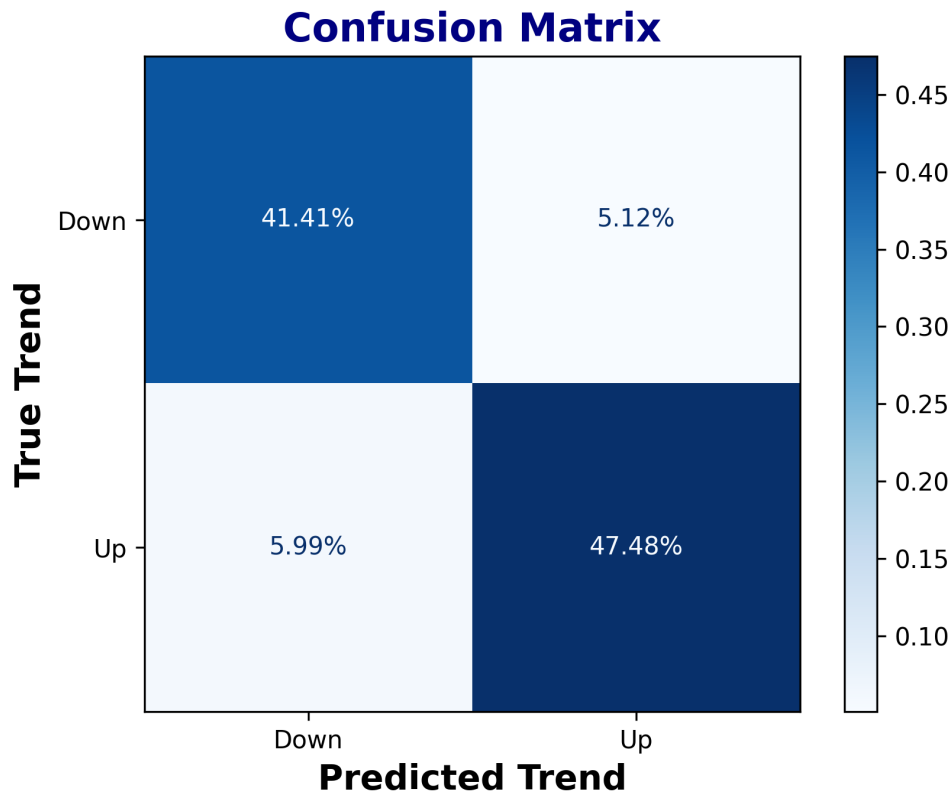
Morgan Stanley

Figure 4.9: Trend confusion matrix of Morgan Stanley with total accuracy of 88.89%.

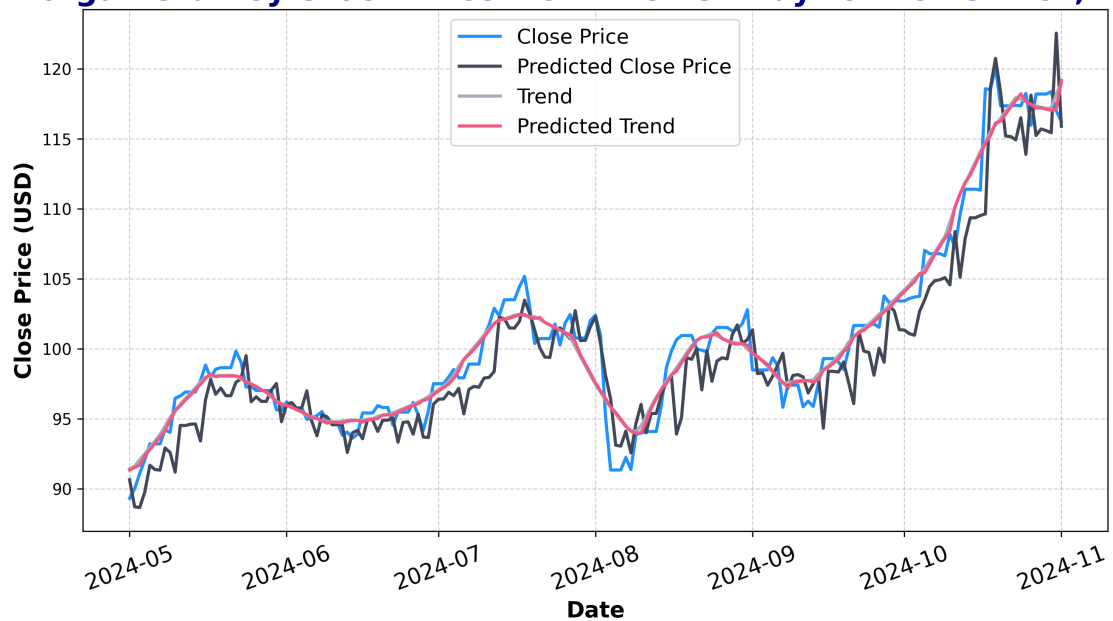
Morgan Stanley Stock Price from the 1st May to the 1st Nov, 2024

Figure 4.10: Morgan Stanley Stock Price from the 1st May to the 1st Nov, 2024.

Goldman Sachs

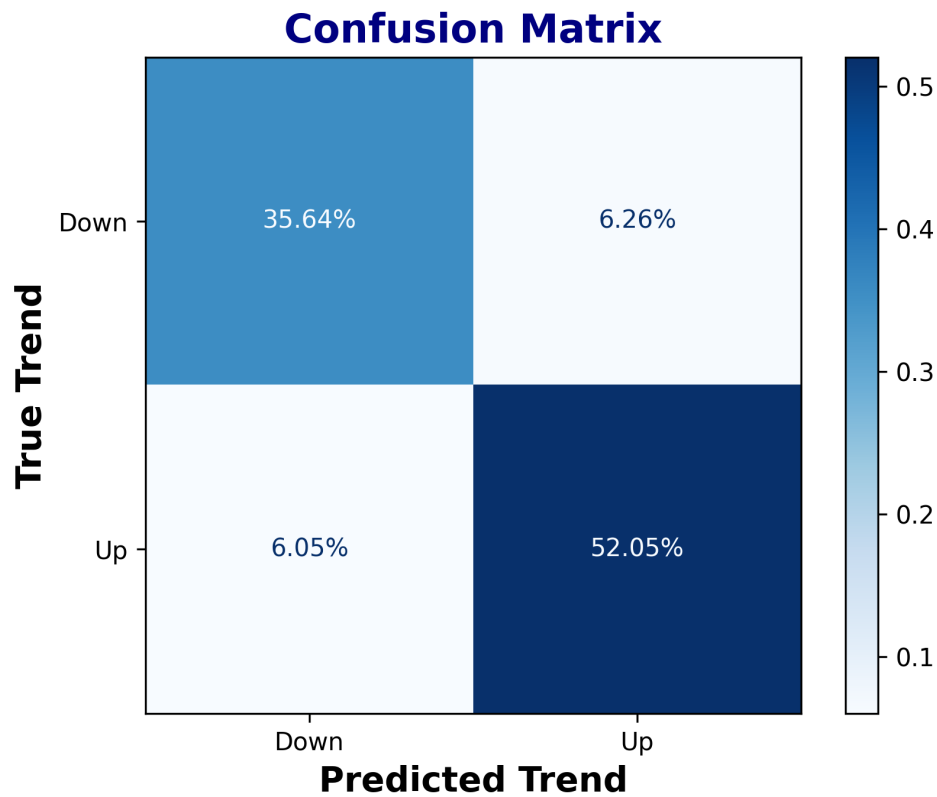


Figure 4.11: Trend confusion matrix of Goldman Sachs with total accuracy of 87.69%.

Goldman Sachs Stock Price from the 1st May to the 1st Nov, 20

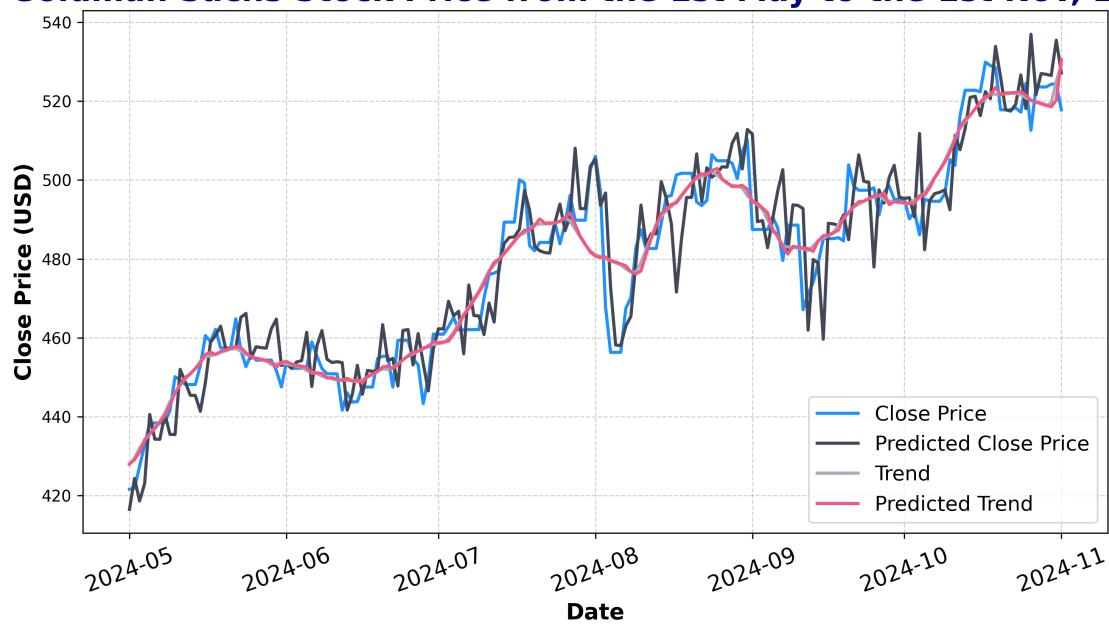


Figure 4.12: Goldman Sachs Stock Price from the 1st May to the 1st Nov, 2024.

Chapter 5

Summary

In this thesis, we investigated the application of deep learning for stock price prediction, which is a challenging time series forecasting problem in the financial industry. We studied different markets by selecting correlation features to enhance the accuracy in predicting multiple stock prices, and we developed a novel neural network architecture that combines the Encoder of the Transformer model with the Time2Vec architecture. Through careful testing and analysis, we found that our model performs exceptionally well, better than standard models such as LSTM and RNN and encoding methods like positional encoding from Transformer; in most cases, our multi-feature proposed model is better than a single-feature model. This success is because our model looks at the bigger picture, using a mix of different features that work together seamlessly, setting a new standard for how good predictions can be in our field. These findings show that using advanced methods for predicting can make a big difference. They also remind us how important it is to think about things like how complicated our model is and which features we use. By paying attention to these details, our study shows us a way to make predictions that work well in real-life situations. In the future, people can try more variations of Time2Vec and Encoder to expect a higher accuracy. We believe that scaling up the dataset (e.g., having the historical data by hour or minute) will allow the model to perform much better.

In summary, our thesis contributes to the ongoing discourse on the application of deep learning in finance, showcasing the effectiveness of correlation-based features and innovative neural network architectures in improving the accuracy and robustness of stock price prediction models. These advancements hold the promise of facilitating more informed investment decisions and enhancing risk management practices in the ever-evolving landscape of financial markets.

Acknowledgements

We would like to thank our supervisor, Professor Dr. Gergő Bognár. Your guidance and support have played an important role in the successful completion of this thesis. Your insightful feedback and suggestions have helped us professionally improve the quality of our thesis. Thank you for your encouragement throughout this journey. Your help has been invaluable, we are truly grateful for the time and effort you have invested in helping us reach to finish this thesis. Once again, thank you for being a mentor and for believing in our abilities. It has been a privilege to work under your supervision.

We would like to thank our friend, Ho Minh Tri, for his assistance during the model training process. Your resources were instrumental in helping us complete the project on time. Thank you for your generosity and dedication.

Bibliography

- [1] Lasse Heje Pedersen. *Efficiently Inefficient: How Smart Money Invests and Market Prices Are Determined*. Princeton University Press, 2019.
- [2] Jr. Gardner Everette S. and Ed McKenzie. “Forecasting Trends in Time Series”. In: *Management Science* 31.10 (Oct. 1985), pp. 1237–1246. URL: <http://www.jstor.org/stable/2631439>.
- [3] Artemios-Anargyros Semenoglou et al. “Investigating the accuracy of cross-learning time series forecasting methods”. In: *International Journal of Forecasting* 37.3 (2021), pp. 1072–1084. ISSN: 0169-2070. DOI: <https://doi.org/10.1016/j.ijforecast.2020.11.009>. URL: <https://www.sciencedirect.com/science/article/pii/S0169207020301850>.
- [4] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-Term Memory”. In: *Neural Computation* 9.8 (1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.
- [5] Shiyang Li et al. “Enhancing the Locality and Breaking the Memory Bottleneck of Transformer on Time Series Forecasting”. In: *CoRR* abs/1907.00235 (2019). arXiv: 1907.00235. URL: <http://arxiv.org/abs/1907.00235>.
- [6] Ashish Vaswani et al. “Attention Is All You Need”. In: *CoRR* abs/1706.03762 (2017). arXiv: 1706.03762. URL: <http://arxiv.org/abs/1706.03762>.
- [7] Aditya Jaiswal Himanshu Gupta. *A Study on Stock Forecasting Using Deep Learning and Statistical Models*. 2024. arXiv: 1512.03385 [cs.CV]. URL: <https://arxiv.org/html/2402.06689v1>.
- [8] Yongsheng Qiao Risheng Qiao Weike Chen. *Prediction of stock return by LSTM neural network*. 2022. arXiv: 1512.03385 [cs.CV].

- [9] Yongsheng Qiao Risheng Qiao Weike Chen. “Prediction of stock return by LSTM neural network”. In: *The Annals of Mathematical Statistics* 22.3 (1951), pp. 400–407. DOI: 10.1214/aoms/1177729586. URL: <https://www.tandfonline.com/journals/uaai20>.
- [10] Souhaib Ben Taieb, Antti Sorjamaa, and Gianluca Bontempi. “Multiple-output modeling for multi-step-ahead time series forecasting”. In: *Neurocomputing* 73 (June 2010), pp. 1950–1957. DOI: 10.1016/j.neucom.2009.11.030.
- [11] Massimiliano Marcellino, James H. Stock, and Mark W. Watson. “A Comparison of Direct and Iterated Multistep AR Methods for Forecasting Macroeconomic Time Series”. In: *Journal of Econometrics* 135.1-2 (2006), pp. 499–526. DOI: 10.1016/j.jeconom.2005.07.013.
- [12] Yongsheng Qiao Risheng Qiao Weike Chen. “Time Series Analysis: Forecasting and Control”. In: *JOURNAL OF TIME SERIES ANALYSIS* 22.3 (2015), pp. 400–407.
- [13] Victor Zhou. *Neural Networks From Scratch*. Accessed: 2024-05-10. URL: <https://victorzhou.com/series/neural-networks-from-scratch/>.
- [14] Xavier Glorot, Antoine Bordes, and Y. Bengio. “Deep Sparse Rectifier Neural Networks”. In: vol. 15. Jan. 2010.
- [15] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. “Learning representations by back-propagating errors”. In: *Nature* 323.6088 (1986), pp. 533–536. ISSN: 1476-4687. DOI: 10.1038/323533a0. URL: <https://doi.org/10.1038/323533a0>.
- [16] Tianyu Wang et al. “From model-driven to data-driven: A review of hysteresis modeling in structural and mechanical systems”. In: *Mechanical Systems and Signal Processing* 204 (2023), p. 110785. ISSN: 0888-3270. DOI: <https://doi.org/10.1016/j.ymssp.2023.110785>. URL: <https://www.sciencedirect.com/science/article/pii/S0888327023006933>.
- [17] Daniel Voigt Godoy. *LeNet Architecture*. Accessed: 2024-11-08. 2021. URL: <https://github.com/dvgodoy/dl-visuals/>.
- [18] Christopher Olah. *Understanding LSTM Networks*. Accessed: 2024-05-10. URL: <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>.

- [19] Jia Wang et al. “CLVSA: A Convolutional LSTM Based Variational Sequence-to-Sequence Model with Attention for Predicting Trends of Financial Markets”. In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*. IJCAI-2019. International Joint Conferences on Artificial Intelligence Organization, Aug. 2019. DOI: 10.24963/ijcai.2019/514. URL: <http://dx.doi.org/10.24963/ijcai.2019/514>.
- [20] Fuli Feng et al. *Enhancing Stock Movement Prediction with Adversarial Training*. 2019. arXiv: 1810.09936 [q-fin.TR].
- [21] Herbert Robbins and Sutton Monro. “A Stochastic Approximation Method”. In: *The Annals of Mathematical Statistics* 22.3 (1951), pp. 400–407. DOI: 10.1214/aoms/1177729586. URL: <https://doi.org/10.1214/aoms/1177729586>.
- [22] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. “Learning representations by back-propagating errors”. In: *Nature* 323.6088 (Oct. 1986), pp. 533–536. DOI: 10.1038/323533a0. URL: <https://doi.org/10.1038/323533a0>.
- [23] John Duchi, Elad Hazan, and Yoram Singer. “Adaptive Subgradient Methods for Online Learning and Stochastic Optimization”. In: *Journal of Machine Learning Research* 12 (July 2011), pp. 2121–2159.
- [24] Seyed Mehran Kazemi et al. “Time2Vec: Learning a Vector Representation of Time”. In: *CoRR* abs/1907.05321 (2019). arXiv: 1907.05321. URL: <http://arxiv.org/abs/1907.05321>.
- [25] Jacob Devlin et al. “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Ed. by Jill Burstein, Christy Doran, and Thamar Solorio. Minneapolis, Minnesota: Association for Computational Linguistics, June 2019, pp. 4171–4186. DOI: 10.18653/v1/N19-1423. URL: <https://aclanthology.org/N19-1423>.
- [26] Qingsong Wen et al. *Transformers in Time Series: A Survey*. 2023. arXiv: 2202.07125 [cs.LG].
- [27] Jian-Hang Li et al. “Multi-Head Attention-Based Hybrid Deep Neural Network for Aeroengine Risk Assessment”. In: *IEEE Access* 11 (2023), pp. 113376–113389. DOI: 10.1109/ACCESS.2023.3323843.

- [28] Mandella Ali M. Fargalla et al. “TimeNet: Time2Vec attention-based CNN-BiGRU neural network for predicting production in shale and sandstone gas reservoirs”. In: *Energy* 290 (2024), p. 130184. ISSN: 0360-5442. DOI: <https://doi.org/10.1016/j.energy.2023.130184>. URL: <https://www.sciencedirect.com/science/article/pii/S0360544223035788>.
- [29] Shunrong Shen, Haomiao Jiang, and Tongda Zhang. “Stock Market Forecasting Using Machine Learning Algorithms”. In: 2012. URL: <https://api.semanticscholar.org/CorpusID:16643114>.
- [30] *Yahoo Finance*. Accessed: 2024-05-05. URL: <https://www.finance.yahoo.com>.
- [31] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV]. URL: <https://arxiv.org/abs/1512.03385>.

List of Figures

2.1	Example for Neural Network.	5
2.2	Example for Recurrent Neural Network	6
2.3	Example for Convolutional Neural Network.	7
2.4	Example for LSTM architecture.	7
2.5	Transformer architecture.	14
2.6	Example for GMNN.	16
3.1	Correlation with NASDAQ being base market.	19
3.2	Correlation with Exxon Mobil being base market.	20
3.3	Correlation with Morgan Stanley being base market.	20
3.4	Preprocessing data pipeline.	22
3.5	Postprocessing pipeline.	24
3.6	Proposed model.	26
4.1	Trend confusion matrix of NASDAQ.	34
4.2	NASDAQ Stock Price from the 1st May to the 1st Nov, 2024.	34
4.3	Trend confusion matrix of S&P500.	35
4.4	S&P500 Stock Price from the 1st May to the 1st Nov, 2024.	35
4.5	Trend confusion matrix of Exxon Mobil.	36
4.6	Exxon Mobil Stock Price from the 1st May to the 1st Nov, 2024.	36
4.7	Trend confusion matrix of Chervon.	37
4.8	Chervon Stock Price from the 1st May to the 1st Nov, 2024.	37
4.9	Trend confusion matrix of Morgan Stanley.	38
4.10	Morgan Stanley Stock Price from the 1st May to the 1st Nov, 2024.	38
4.11	Trend confusion matrix of Goldman Sachs.	39
4.12	Goldman Sachs Stock Price from the 1st May to the 1st Nov, 2024.	39

List of Tables

3.1	Data from Yahoo Finance.	21
3.2	Data after apply Moving Average.	22
3.3	Data after apply Percentage Change.	23
3.4	Data after apply Normalization.	23
3.5	Data after apply GMNN.	24
3.6	Suggested values for hyperparameters.	27
3.7	Hyperparameters Tuning.	29
4.1	Evaluations by target of each group.	30
4.2	Comparing to SOTA models – Group A.	31
4.3	Comparing to SOTA models – Group B.	32
4.4	Comparing to SOTA models – Group C.	32