Galformer: A Transformer with Generative Decoding and a Hybrid Loss Function for Multi-Step Stock Market Index Prediction

Yi Ji[a]: Conceptualization, Methodology; **Yuxuan Luo**[a,*]: Writing, Formal analysis, Validation; **Aixia Lu**[a]: Software; **Duanyang Xia**[a]: Visualization; **Lixia Yang**[b]: Supervision; **Alan Wee-Chung Liew**[c]: Supervision.

[a] School of Electrical Information Engineering, Jiangsu University, 301 Xuefu Road, Zhenjiang, 212013, Jiangsu, China.
[b] School of Electronic Information Engineering, Anhui University, 111 Jiulong Road, Hefei, 610101, Anhui, China.
[c] School of Information and Communication Technology, Griffith University, 170 Kessels Road, Nathan, 4111, Queensland, Australia.

[*] Corresponding author. E-mail address: 2222107005@stmail.ujs.edu.cn (Yuxuan Luo).

Abstract:
The prediction of stock market fluctuations is crucial for decision-making in various financial fields. Deep learning algorithms have demonstrated outstanding performance in stock market index prediction. Recent research has also highlighted the potential of the Transformer model in enhancing prediction accuracy. However, the Transformer faces challenges in multi-step stock market forecasting, including limitations in inference speed for long sequence prediction and the inadequacy of traditional loss functions to capture the characteristics of noisy, nonlinear stock history data. To address these issues, we introduce an innovative transformer-based model with generative decoding and a hybrid loss function, named "Galformer," tailored for the multi-step prediction of stock market indices. Galformer possesses two distinctive characteristics: (1) a novel generative style decoder that predicts long time-series sequences in a single forward operation, significantly boosting the speed of predicting long sequences; (2) a novel loss function that combines quantitative error and trend accuracy of the predicted results, providing feedback and optimizing the transformer-based model. Experimental results on four typical stock market indices, namely the CSI 300 Index, S&P 500 Index, Dow Jones Industrial Average Index (DJI), and Nasdaq Composite Index (IXIC), affirm that Galformer outperforms other classical methods, effectively optimizing the Transformer model for stock market prediction.

Keywords:
Stock index prediction; Transformer; Galformer; Deep learning.

1. Introduction

Stock market prediction, as a focal point within the financial markets, explores the external dynamics that govern the complex interactions of significant wealth. These markets exhibit constant fluctuations, which bear substantial implications, capable of influencing the stability of financial systems, as exemplified by the notable 2008 financial crisis[1]. Throughout the extensive timeline of capital markets, numerous prediction methodologies have emerged, encompassing technical analysis, fundamental analysis, and time series analysis[2]. In the current landscape, marked by rapid advancements in computer science, especially in the field of deep learning, researchers are optimistic about harnessing vast data repositories to comprehend the intricate dynamics that underlie stock prices[3]. Financial institutions are increasingly adopting deep learning algorithms for making investment decisions, aiming to minimize subjective human biases. As a result, the integration of deep learning techniques with financial time series forecasting has become an alluring frontier of research, capturing the inherent complexities intrinsic to financial markets[4].

Several studies in the field have demonstrated the effectiveness of deep learning frameworks in predicting financial market trends. Compared to traditional machine learning algorithms, Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), and Long Short-term Memory Neural Network (LSTM) structures have demonstrated superior predictive capabilities[5]. In the study Selvin conducted, CNN, RNN, and LSTM models were compared for stock prediction, with LSTM proving to be the most accurate due to its ability to retain long-term memory for stock sequences[6]. Bao proposed a combined approach of Autoencoder and LSTM for stock price prediction, which yielded improved accuracy and profitability compared to using a single structure[7]. Furthermore, researchers have explored other advanced deep learning frameworks for stock prediction, including generative adversarial networks and reinforcement learning[8],[9]. Jiang provided a comprehensive overview of recent advancements in applying deep learning to stock market prediction[10].

Despite the extensive adoption of deep learning in predicting stock market trends, conventional CNN and RNN architectures still exhibit inherent limitations. CNN's pooling layers, for example, can result in information loss as they disregard the relationships between parts and wholes[11]. Similarly, RNN is prone to issues like gradient disappearance and explosion during the back-propagation process[12]. To overcome these limitations, Vaswani proposed an innovative deep learning architecture called Transformer[13]. In contrast to traditional CNN and RNN frameworks, Transformer utilizes the attention mechanism, enabling it to achieve remarkable success in natural language processing (NLP) tasks. One notable advantage of Transformer is its ability to capture global information through parallel training, distinguishing it from the sequential nature of RNN and LSTM. Recently, Transformer has found wide-ranging applications in domains such as computer vision[14], audio processin[15], and chemical synthesis[16].

Taking inspiration from the accomplishments of the Transformer model in handling sequential data for natural language processing tasks, it is logical to explore its application in stock market prediction. Nevertheless, the predominant emphasis in current research lies in leveraging Transformer to analyze textual data like financial news and social media comments, aiming to predict price trends[17]. Recently, Zhou proposed a highly influential long-term sequence prediction model based on the improved Transformer, called Informer, which has achieved excellent prediction effects on data sets in various fields, driving the research enthusiasm in this direction[18]. Shortly thereafter, at NeurIPS 2021, Wu introduced the Autoformer model, which integrated their previous work with the Informer model design[19]. This new model significantly outperformed previous models on the same prediction tasks, achieving a 38% relative improvement in performance. However, there is a scarcity of studies that directly employ Transformer to forecast historical time series data in the stock market. What's more, the training on dozens of GPUs and expensive deploying cost make the Vanilla Transformer model unaffordable on real-world stock market prediction[20]. The efficiency of the self-attention mechanism and Transformer architecture becomes the bottleneck of applying them to stock market prediction. Furthermore, our experiments demonstrate that there is room for improvement in the trend prediction accuracy of Vanilla Transformers when applied to stock market data. In order to enhance the prediction capacity, we address these limitations and achieve significant improvements in the proposed Galformer model, surpassing mere efficiency considerations.

The recursive multi-step prediction in Transformer models leads to the accumulation of errors at each recursive step. Stock market forecasting requires processing vast amounts of data, and traditional sequential models (such as RNNs) are inefficient for long sequence multi-step predictions due to their inability to utilize parallel computation. These specific challenges in multi-step stock market forecasting motivated the development of the Galformer model.

To this end, our study focuses explicitly on addressing these challenges. We improve the structure of the Transformer decoder, apply a novel kind of loss function and conduct extensive experiments.

The contributions of this paper are as follows:

(1) We suggest using the Transformer framework to forecast stock market indices, which directly models the daily adjusted closing price data, focusing on the specific values of the stock index in the future.
(2) We introduce a novel generative style decoder to avoid cumulative error spreading during the inference phase and reduce computational complexity.
(3) We propose a novel loss function to enhance the final trend prediction accuracy of the forecasting model.

In this study, prominent global stock market indices were selected as experimental objects, such as the CSI 300 Index, S&P 500 Index, Dow Jones Industrial Average Index, and Nasdaq Composite Index. The historical datasets of each stock index are divided into training, validation, and testing sets in a ratio of 7:2:1. The training and validation sets are employed for train model parameters and uncover hidden patterns, while the testing set is used to evaluate model performance and make comparisons. Our methodology involves preprocessing the dataset to enhance model convergence. We provide input data to the improved Transformer model, including the daily adjusted closing prices from the preceding days. Utilizing the encoder-decoder architecture, the Transformer generates predictions for the closing price of the subsequent day, which are subsequently assessed against the actual values. Performance evaluation includes metrics such as prediction accuracy and the effectiveness of trading strategies. Through comprehensive experiments, we demonstrate that Galformer outperforms ARIMA, LSTM, Transformer, and Informer models in terms of prediction accuracy (measured by the mean squared error) and trend prediction accuracy. These outcomes highlight the superior performance of the Transformer in stock market prediction, establishing it as a more effective alternative.

This paper is organized as follows: Section 2 provides a review of related work about applying deep neural networks to stock data and loss functions. Section 3 shows the related research theories used in the experiments and the proposed model. Section 4 illustrates the experimental process of the proposed model and the analysis of the predicted results. Finally, Section 5 concludes this paper.

2. Related work

2.1. Feasibility of Transformer prediction

Anticipating future outcomes by leveraging historical and current data is the essence of prediction. In the context of time series analysis, if we assume the presence of a chronological sequence x_{t-15}, x_{t-14}, …, x_{t-1}, The objective is to forecast future sequence values x_t, x_{t+1}, x_{t+2} based on the known sequence[21].

According to technical analysis, the price movement of stocks encompasses all the information regarding the stock market. By directly modeling and predicting stock prices, deep learning techniques can effectively capture the underlying patterns from historical data, enabling efficient forecasting of future values[22].

During initial research, RNN and LSTM emerged as common frameworks employed for the prediction of financial time series data. The utilization of LSTM, in conjunction with optimization algorithms like Particle Swarm Optimization (PSO), has demonstrated its efficacy in achieving favorable predictive outcomes for stock data[23].

The groundbreaking success of the Transformer model in solving NLP problems was evident upon its introduction. From then on, many researchers started trying to apply the Transformer model to time series prediction. Among them, a highly influential long-term sequence prediction model based on the improved Transformer, called Informer, was introduced by Zhou[18]. Based on these research advancements, this paper enhances the Transformer as an innovative approach for predicting stock market indices.

2.2. Loss functions

In neural networks, the loss function plays a crucial role in measuring the discrepancy between predicted outputs and the true values. It serves as a guide for the network to adjust its parameters during the training process. Various loss functions have been developed to cater to different types of prediction tasks[24].

One commonly used loss function is the mean squared error (MSE), which calculates the average squared difference between predicted and true values. It is widely applied in regression problems, where the goal is to minimize the overall squared error. In neural network models for time series prediction, the MSE loss function has been widely employed, in contrast to the Binary Cross-Entropy Loss function commonly used in classification tasks[25]. For instance, the novel time series model called Informer, proposed in 2021, utilizes the MSE loss function. Furthermore, in the article by Ji that employs LSTM for stock market prediction, the loss function used is also MSE[26].

In some cases, specialized loss functions are designed to address specific requirements. For instance, the focal loss function is used in object detection tasks to handle the class imbalance problem, giving more weight to hard-to-classify examples.

Furthermore, there are loss functions tailored for sequence prediction tasks. The sequence-to-sequence models often use the connectionist temporal classification (CTC) loss, which enables training without requiring aligned input-output pairs. This loss function allows the network to learn to align the input sequence with the corresponding output sequence.

Given that stock market index prediction is a time series forecasting task, we propose to improve our model's performance by enhancing the MSE loss function. Experimental results indicate that there is indeed room for improvement in the trend accuracy of stock market index prediction models using the MSE loss function.

3. Methodology

3.1. Traditional Transformer model

3.1.1. Encoder-decoder Architecture

Nowadays, the encoder-decoder architecture is widely employed in machine translation issues, and the Transformer is a prominent example[27]. The encoder encodes the input $X^t = \{x_1^t, x_2^t, \cdots, x_{L_x}^t\}$ into a hidden state $H^t = \{h_1^t, h_2^t, \cdots, h_{L_h}^t\}$, and the decoder decodes the hidden state H^t to generate the output $Y^t = \{y_1^t, y_2^t, \cdots, y_{L_y}^t\}$. During this iterative process of dynamic decoding, the decoder computes a new hidden state h_{k+1}^t based on the previous state h_k^t, along with other necessary outputs at step k[28]. Then, the predicted value y_{k+1}^t for the $(k+1)th$ sequence is obtained. It is evident that the encoder-decoder architecture presents a method for handling long sequence data, making it applicable for long-term time series forecasting.

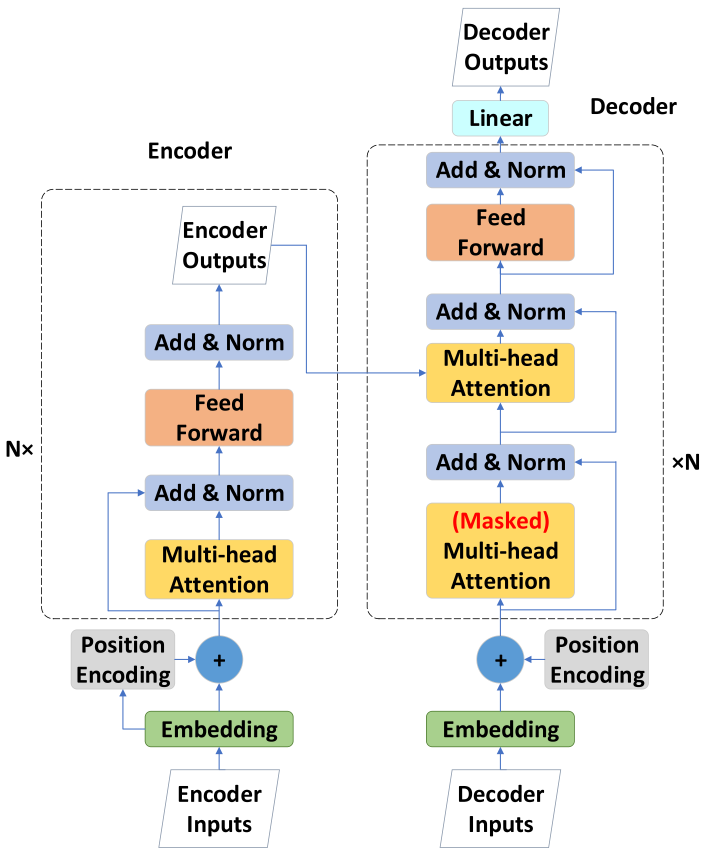


Fig. 1. Transformer model overview.

The structure of the traditional Transformer model is illustrated in Fig. 1. On the left-hand side of Fig. 1, the encoder consists of *N* identical layers. Each layer comprises two sub-layers: a multi-head self-attention layer and a fully connected neural network. Residual connections and normalization are employed between each sub-layer to enhance performance. The decoder on the right-hand side of Fig. 1 is similar to the encoder, with the addition of one more multi-head self-attention layers compared to the encoder. It is worth noting that in this study, the traditional Transformer model is applied to financial time series prediction, resulting in modifications to the inputs of both the encoder and decoder to accommodate processed historical stock data sequences. Additionally, the masked attention mechanism highlighted in red in Fig. 1 is not utilized in our experimental model, as all inputs to the decoder consist of known historical data and do not include future information.

3.1.2 Embedding and positional encoding

Embedding is a frequently employed data preprocessing technique in the realm of neural network predictions for natural language processing (NLP) tasks. It facilitates the provision of more detailed feature information from the original data by extending its dimensions[29]. For instance, in this study, the encoder inputs, which are the historical price data of the previous 20 days denoted as $X = \{x_t : 1, \cdots 20\} \in R^{20}$, undergo a linear embedding layer of the same 512 dimensions as the vanilla Transformer. This transformation results in multi-dimensional feature data of dimension $R^{20\times512}$, facilitating subsequent processes.

In the context of Transformers, to capture the sequential information within time series, positional encodings are introduced into the embedded inputs. Various frequencies of sine and cosine functions are utilized for encoding the positional information:

$$\mathrm{PE}_{(t,2s)} = sin\left(t \,/\, 10000^{2s/d}\right) \tag{1}$$

$$\mathrm{PE}_{(t,2s+1)} = cos\left(t \,/\, 10000^{2s/d}\right) \tag{2}$$

where d refers to the dimensionality of the embedding layer which is the aforementioned 512, $1 \leq 2s \leq d$. Subsequently, the embedded input and positional encoding are concatenated and fed into the encoder layers.

3.1.3 Self-attention

The attention mechanism focuses a limited attention on crucial local regions, conserving computational resources and rapidly acquiring the most valuable information.

The self-attention mechanism in vanilla Transformer is defined as follows:

$$Attention(Q,K,V) = softmax\left(\frac{\mathrm{QK^T}}{\sqrt{d}}\right)\mathrm{V} \tag{3}$$

where *Q*, *K* and *V* are query, key and value matrices respectively. From Fig. 2, it can be observed that *Q*, *K*, and *V* are the outputs obtained by multiplying the same input matrix with three different matrices. This can also be comprehended as the same input being processed through three distinct linear layers to yield the outputs.

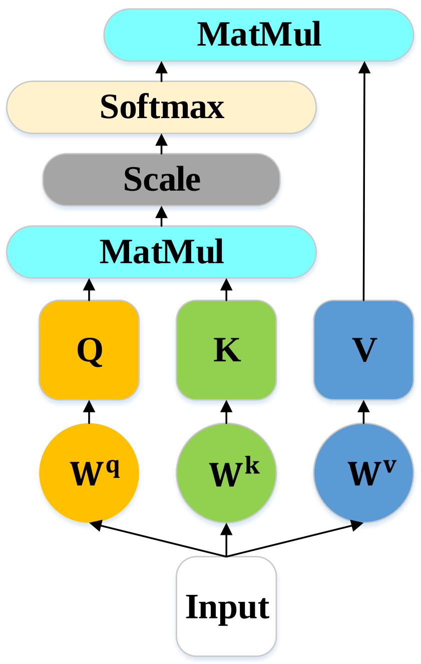


Fig. 2. Self-attention mechanism overview.

For example, in this experiment, the encoder's input consisting of historical data from the past 20 days undergoes embedding and position encoding before undergoing self-attention mechanism computations, resulting in data reconstruction. The input, originally comprised of mutually independent data from the 20 days, is reconstructed based on attention weights that can be seen as capturing inter-day relationships or correlations. The output for each day can be viewed as an aggregation of the 20 days' data, weighted and aggregated according to attention scores obtained from the *softmax* function. In essence, Multi-head attention can be understood as multiple self-attention mechanisms working in parallel.

3.1.4 Multi-head attention

In essence, Multi-head attention can be considered as the utilization of multiple self-attention mechanisms, thereby achieving improved performance. In the multi-head attention mechanism, each attention function is executed in parallel with the corresponding projected versions of the query, key, and value matrices. The outputs of all attention functions are then concatenated together, generating the final result through a linear layer. The equation representation of multi-head attention is as follows:

$$MultiHead(Q,K,V) = Concat(head_1, head_2, \dots, head_h)W^O \tag{4}$$

$$head_i = Attention\left(QW_i^Q, KW_i^K, VW_i^V\right) \tag{5}$$

where $i = 1, \dots, h$ and W_i^Q, W_i^K, W_i^V, W^O are corresponding weight matrices.

The multi-head attention achieves the effect of preventing overfitting by running multiple self-attention mechanisms in parallel and then integrating their outputs. This technique is often employed in neural network forecasting research to enhance model performance.

3.2. Galformer model

3.2.1. Generative decoder

The standard decoder structure in Fig. 1 consists of two identical multi-head attention layers. However, in the case of the vanilla Transformer decoder, it employs a teacher-forcing mechanism to iteratively generate output predictions[30]. This generative inference approach results in a drastic computational slowdown when the model is engaged in long-term forecasting. Consequently, we have enhanced the conventional recursive prediction method by introducing a forward process for generating extended sequence outputs, addressing the limitation of the traditional transformer model in inference speed.

The teacher-forcing mode is characterized by using the ground truth as input during training, rather than using model predictions. Its advantage lies in the parallel generation of prediction results during training, which enhances training efficiency. However, its drawback is that during testing and actual predictions, due to the inability to anticipate future data, sequential recursive predictions are still required, leading to a decrease in efficiency.

Therefore, we have improved the decoder component by discarding the teacher-forcing mechanism and adopting a novel one-step decoder. This new approach allows for the parallel generation of prediction results in both training and testing processes, enhancing operational efficiency. The testing procedures employing teacher-forcing and the new decoder are illustrated in Fig. 3 and Fig. 4, respectively. Where, y represents the true values, $\hat{y}$ represents the predicted values, t represents the first day to be predicted, and n represents the total number of days to be predicted, respectively. The blue blocks in the figure represent the prediction steps of the decoder.

It is obvious that when the teacher-forcing structure is used for n-step ahead prediction, after the first prediction, an additional n-1 iterations of recursive forecasting are required. This significantly increases the computational time complexity. Meanwhile, for data with high precision requirements, such as financial data, iterative predictions will inevitably lead to the accumulation of prediction errors. The main reason for these issues arises from feeding future data into the decoder during model training, where the masking mechanism limits each operation to predict only a single-step data. In contrast, the proposed generative decoding approach modifies the input to the decoder to the end of the historical data. Choosing the end of the historical sequence is because it is closer to the sequence to be predicted. Through subsequent operations such as position encoding, it can contribute more to the final prediction result that is closer to the sequence to be predicted. Moreover, the dimensionality of the end of the historical sequence input to the decoder is consistent with the sequence to be predicted that the decoder outputs.

Overall, the proposed generative decoding approach fundamentally eliminates the input of future data by changing the data structure of the decoder's input and output compared to the traditional Transformer. This ultimately avoids the accumulation of errors caused by recursive predictions. Additionally, predicting multiple steps in a single operation simplifies the testing process and reduces computational complexity.

The process of teacher-forcing decoding and generative decoding methods, as well as the differences in input and output during model training and testing periods, can be observed in Fig. 5, 6, 7, and 8 when taking the historical data of the previous 20 days as an example to predict future data for the following 3 days. Here, *Day[1-20]* represents the sequence of true values for the entire 20 days, *Day[t]* and *Day[t*]* represent the actual value and predicted value of day *t*, respectively.

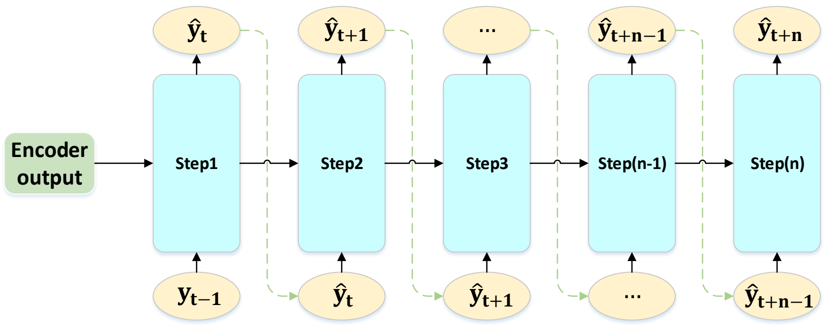


Fig. 3. Testing procedure employing teacher-forcing.

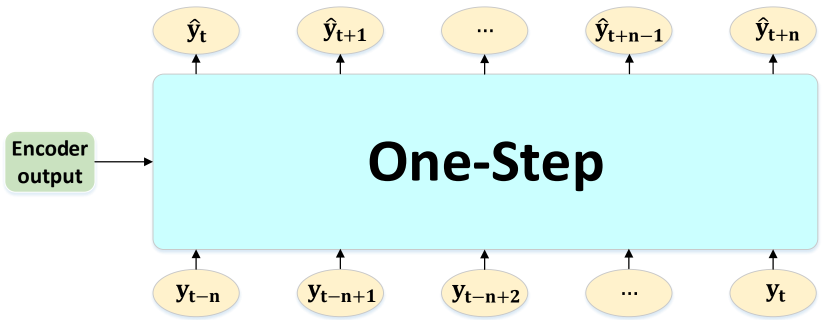


Fig. 4. Testing procedure employing the proposed generative decoding.

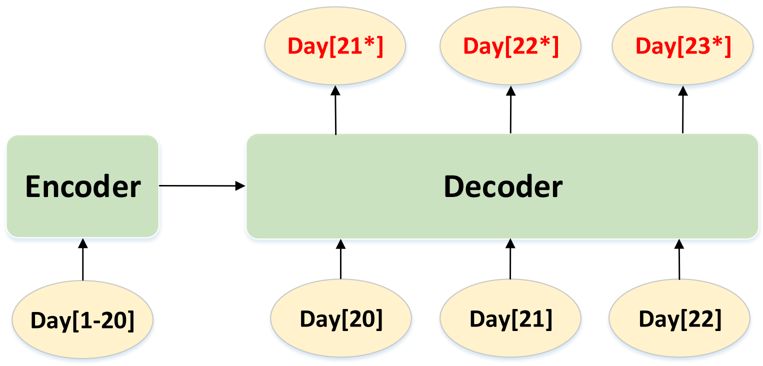


Fig. 5. Training procedure of the teacher-forcing decoding.

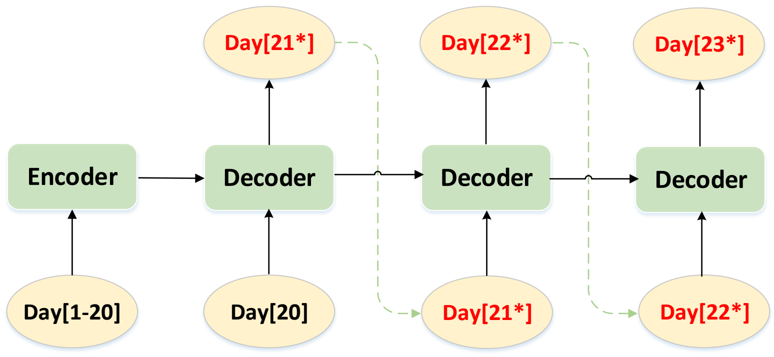


Fig. 6. Testing procedure of the teacher-forcing decoding.

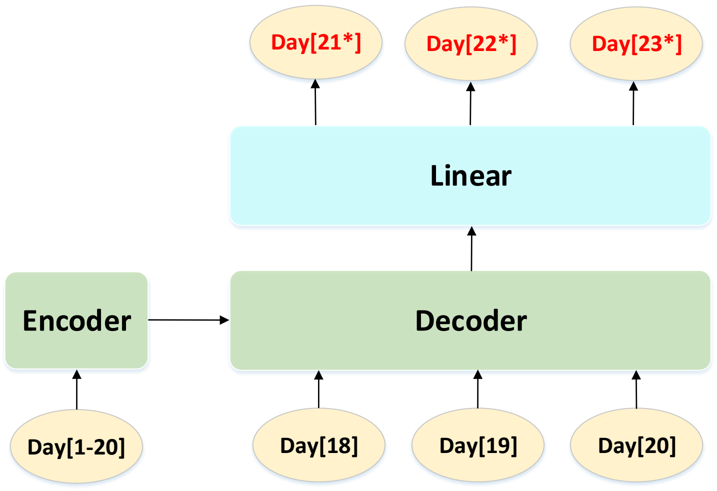


Fig. 7. Training and testing procedure of the proposed generative decoding. The training and testing procedure remains consistent, with the decoder strictly limited to present data. The decoder outputs the predicted value for the next three days in one step through a linear layer of dimension 3.

3.2.2 Hybrid Loss function

The loss function has been discussed in Section 2.2 of this paper. In the context of neural network applications for time series regression prediction, the preferred loss function is MSE, similar to how Binary Cross-Entropy Loss is commonly used in most neural network applications for classification. MSE is employed to quantify the average squared error between the model's predictions and the actual values. Its value serves as feedback on the quality of the model's training results, aiding in model optimization. Specifically, the equation for calculating MSE is specifically outlined in Eq. 6:

$$\text{MSE} = \frac{1}{N}\sum\nolimits_{t=1}^{N}\left(y(t) - \hat{y}(t)\right)^2 \tag{6}$$

where, N stands for the number of samples in the test set, while $y(t)$ denotes the true values in the test set, and $\hat{y}(t)$ signifies the corresponding predicted outcomes.

The MSE loss function is a suitable metric for predicting regular and predictable time series, such as traffic, weather, and electricity data. However, it exhibits limitations when applied to forecasting time series data with significant nonlinear fluctuations, like stock market index data. This is because close numerical values do not necessarily imply correct predictions of the data's upward or downward trends. For stock market data forecasting, the trends in rises and falls are often more critical than the actual numerical values. This is illustrated in Fig. 8:

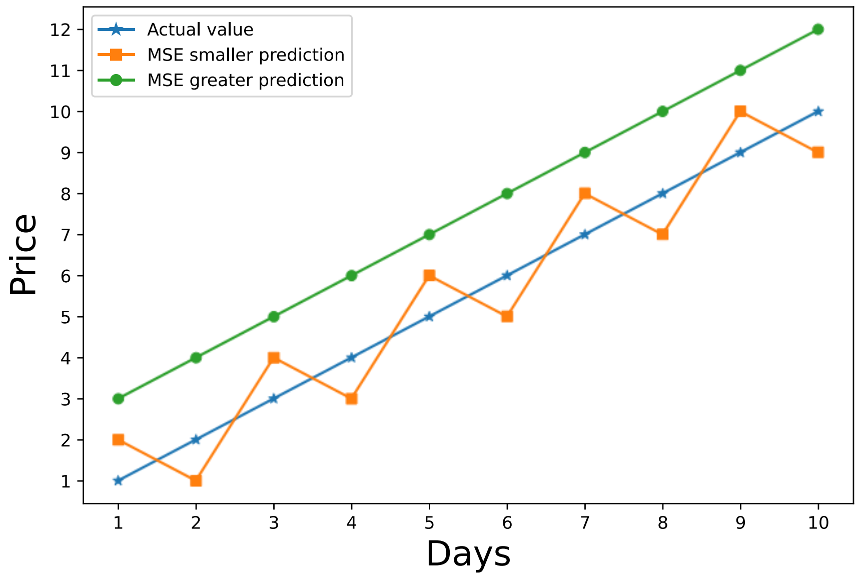


Fig. 8. Comparison of Predictions: MSE Smaller vs. MSE Greater.

Fig. 8 illustrates a significant contrast in MSE values, with the orange line showing an MSE of 1, markedly smaller than the green line's MSE of 4. While the orange line provides more accurate numerical predictions, it is notably inaccurate in forecasting the trends of rise and fall. Conversely, despite the green line exhibiting larger numerical prediction deviations, it excels in accurately predicting trends of rise and fall. This also indicates that a prediction model solely relying on the MSE loss function is not sensitive to the trends of data rises and falls. For the prediction of stock markets with frequent fluctuations in rises and falls, this insensitivity is a significant drawback as it directly impacts the accuracy of stock decisions.

Therefore, in order to solve the limitation of the traditional Transformer model in loss function adequacy, this paper proposes incorporating the accuracy of trend prediction into the loss function for regression prediction. By combining numerical prediction errors and trend prediction errors, it provides a more effective feedback mechanism for training regression prediction neural networks. The newly proposed loss function is detailed in Eq. 7:

$$Loss = MSE + (1 - ACC) \times 10 \wedge floor(lg(abs(MSE))) \tag{7}$$

$$ACC = (TP + TN) / (TP + TN + FP + FN) \tag{8}$$

Where, *ACC* represents trend prediction accuracy, the *floor()* function is used to round the result to the nearest integer that is less than or equal to the original result, $lg\, x$ represents $log_{10}\, x$. As *MSE* and *ACC* are often not on the same order of magnitude, the coefficient multiplied after *(1 – ACC)* is used to scale it to the same order of magnitude as *MSE*, ensuring that both metrics provide the model with feedback of the same magnitude. A larger *MSE* in the prediction results will lead the model to learn in a direction that reduces *MSE* while maintaining a high *ACC*, thus improving numerical accuracy. Conversely, a smaller *ACC* will cause the model to learn in a direction that increases *ACC* while maintaining a low MSE, thereby enhancing trend accuracy. Consequently, to minimize the loss function, the model can balance low *MSE* and high *ACC* during the training process, ensuring both numerical and trend accuracy, ultimately improving the model's overall predictive performance. *TP*(True Postive) represents the number of samples in the dataset where the upward trend between two consecutive days is correctly predicted. Similarly, *TN*(True Negative) represents the number of samples where the downward trend is correctly predicted, *FP*(False Postive) represents the number of samples where the downward trend is incorrectly predicted, and *FN*(False Negative) represents the number of samples where the upward trend is incorrectly predicted.

By incorporating both *MSE* and *ACC*, the hybrid loss function ensures that the model pays attention to both the precise values of the predictions and the overall trend. This dual focus helps the model to accurately capture the inherent volatility and nonlinear patterns in stock market data. Furthermore, *MSE* primarily focuses on minimizing the difference between predicted and actual values, which can sometimes be overly sensitive to noise in the data. By including *ACC* in the loss function, the model also learns to recognize and follow the underlying trend, which helps to mitigate the impact of random noise and short-term fluctuations.

4. Experiments

4.1. Development environment

The development environment of this study is Python 3.10 on a Windows 10 operating system. The entire process is implemented using the TensorFlow machine learning platform, with a TensorFlow version of tensorflow-gpu-2.9.1.

4.2. Dataset and preprocessing

In this paper, we conducted back-testing experiments on several major global stock market indices to demonstrate the performance of Galformer in stock market prediction. Given the significant influence of the United States and China as leading economies worldwide, their stock markets have a substantial impact. Therefore, we selected four stock market indices from these two countries to represent the global market: Shanghai and Shenzhen 300 Index (CSI 300) in China, the Standard & Poor's 500 Index (S&P 500), the Dow Jones Industrial Average Index (DJI), and the Nasdaq Composite Index (IXIC) in the United States. Predicting the movements of these major indices is particularly challenging due to their complex, nonlinear, and volatile nature. Using such challenging datasets helps to rigorously test and demonstrate the effectiveness of the Galformer model in handling complex stock market prediction tasks. We modeled the daily adjusted closing prices of these indices over a 10-year period, spanning from July 1, 2011, to June 29, 2021. The original dataset is initially divided into three parts: the first 70%, the middle 20%, and the final 10%, representing the training set, validation set, and test set, respectively. In the training and testing procedures, the data is divided into groups of 20 days each. The historical data for the encoder is input from each group, with the last three days of each group serving as the historical data input for the decoder. The final prediction result labels correspond to the future data of the last three days of each 20-day period, which constitute the decoder's output sequence.

Due to the potential issue of values in the original dataset having magnitudes that could make prediction challenging, all three datasets need to undergo the Z-Score standardization method, transforming them into time series with a mean of 0 and a standard deviation of 1. The specific expression for this can be found in Eq. 9.

$$x'(t) = \frac{x(t)-\mu}{\sigma} \tag{9}$$

where, t represents each trading day, $x(t)$ represents the original adjusted closing price, $x'(t)$ represents the adjusted closing price after the standardization process, while μ and σ denote the mean and standard deviation of the original adjusted closing prices, respectively.

4.3. Parameter settings of different models

The actual value dataset used for all model inputs consists of the adjusted closing prices known for the previous 20 days, while the predicted label values for the outputs represent the adjusted closing prices for the following 3 days, which are unknown. Parameter settings of the four baselines and Galformer proposed in this study are detailed in Table 1. Where p represents the autoregressive order, d represents the difference order, and q is the moving average order. *Node1* and *Node2* refer to the number of nodes in the first and second LSTM hidden layer, respectively. η represents the learning rate of the Adam optimizer. L_{en} represents the length of the encoder input sequence, while L_{de} represents the length of the decoder input sequence. *d_model* represents the dimensionality of the vector after input data undergoes embedding. *num_head* represents the number of attention heads in multi-head attention, while *num_layer* represents the number of layers in both the encoder and decoder. *U_part* represents the number of valuable queries sampled by *ProbAttention*. *dropout* is the dropout rate for each sub-layer.

TABLE 1. Parameter settings of five models

Model	Parameter Settings
ARIMA	(p, d, q) = (1, 1, 1)
LSTM	*Node1* = 200, *Node2* = 200, *Epoch* = 300, η = 0.003, *Batch_size* = 64, *look-back period* = 20
Transformer	*Epoch* = 300, η = 0.003, *Batch_size* = 64, L_{en} = 20, L_{de} = 3, *d_model* = 512, *dense_dim* = 2048, *num_head* = 8, *num_layer* = 6, *dropout* = 0.1
Informer	*Epoch* = 300, η = 0.003, *Batch_size* = 64, L_{en} = 20, L_{de} = 6, *d_model* = 512, *dense_dim* = 2048, *num_head* = 8, *num_layer* = 6, *U_part* = 5, *dropout* = 0.1
Galformer	*Epoch* = 300, η = 0.003, *Batch_size* = 64, L_{en} = 20, L_{de} = 3, *d_model* = 512, *dense_dim* = 2048, *num_head* = 8, *num_layer* = 6, *dropout* = 0.1

4.4. Evaluation indicators

In this research, we utilize five evaluation indicators to assess the prediction model's performance: trend prediction accuracy (ACC), root mean square error (RMSE), mean absolute percentage error (MAPE), mean absolute error (MAE), and R^2 score (R^2). These metrics are derived by comparing the overall predicted value sequence obtained from the model with the historical true value sequence of the dataset. The numerical results of these metrics are used to evaluate the accuracy of the model's predictions.

ACC represents the accuracy of the model in predicting the daily upward or downward trend of the data, with higher values indicating better model performance. RMSE, MAPE, MAE, and R^2 reflect the numerical differences between the model's predicted values and the true value sequences. Smaller values for the first three metrics imply that the model's predictions are closer to the true values, indicating better predictive performance. The R^2 value ranges between 0 and 1, where 0 signifies completely incorrect predictions and 1 signifies perfect predictions. Therefore, a higher R^2 value reflects better predictive performance of the model.

These evaluation metrics are defined by Eq. 8 and Eq. (10-13).

$$\mathrm{RMSE} = \sqrt{\frac{1}{N}\sum_{t=1}^{N}\left(y(t) - \hat{y}(t)\right)^2} \tag{10}$$

$$\mathrm{MAPE} = \frac{1}{N}\sum_{t=1}^{N}\left|\frac{y(t)-\hat{y}(t)}{y(t)}\right| \times 100\% \tag{11}$$

$$\mathrm{MAE} = \frac{1}{N}\sum_{t=1}^{N}|y(t) - \hat{y}(t)| \tag{12}$$

$$\mathrm{R}^2 = 1 - \frac{MSE\left(y(t)-\hat{y}(t)\right)}{Var\left(y(t)\right)} \tag{13}$$

where N is the sample size in the test dataset. $y(t)$ represents the actual value in the test dataset, while $\hat{y}(t)$ denotes the corresponding predicted value. Var and MSE stand for variance and mean squared error, respectively.

4.5. Computation efficiency

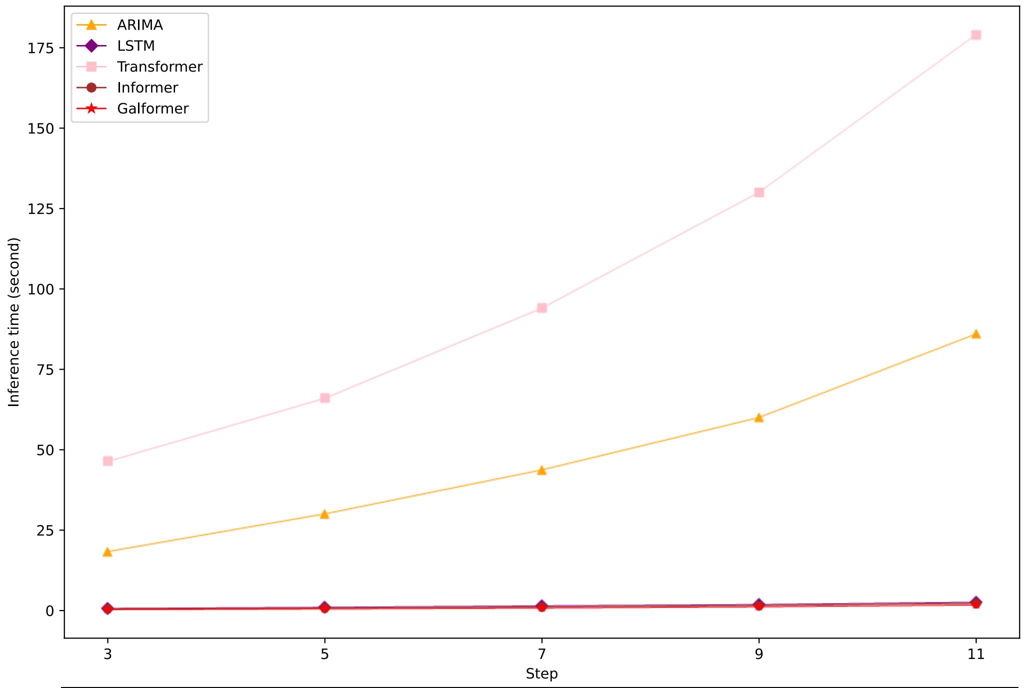
We employed a relatively fair multi-parameter setup and the best current implementations of all methods, comparing the test runtime of various models through experiments with data from the **CSI 300** Index, **S&P 500** Index, **DJI** Index, and **IXIC** Index over the same 10-year period. In testing, our method outperformed other classical prediction methods by a significant margin. Table 2 summarizes the comparison of time complexity in the model testing phase. Assuming the test dataset is in the format of (n, L), where n is the length of the test dataset, L is the length of multi-step predictions, and nL represents the total number of days in the test dataset. The performance of Galformer aligns with our runtime experiments.

With the multivariate setting and the latest implementations of all methods, we conducted a thorough runtime comparison in Fig. 9. Due to the orders of magnitude larger runtimes of Transformer and ARIMA, we present two result figures. One depicts the runtime results for all models, while the other illustrates the runtime results for selected models, including LSTM, Informer, and Galformer. During the testing phase, our methods are much faster than others with the generative style decoding. This demonstrates that the generative-style decoder, by performing parallel operations to output multi-step predictions in a single step, improves the speed of predicting long time-series sequences compared to traditional methods that derive multi-step predictions through recursive single-step forecasting.

TABLE 2. Time complexity of five models

Model	Time Complexity	Average Test Runtime(s)
ARIMA	$O(nL)$	18.26
LSTM	$O(nL)$	0.5630
Transformer	$O(nL)$	46.36
Informer	$O(n)$	0.4862

Model	Time Complexity	Average Test Runtime(s)
Galformer	$O(n)$	0.3411*



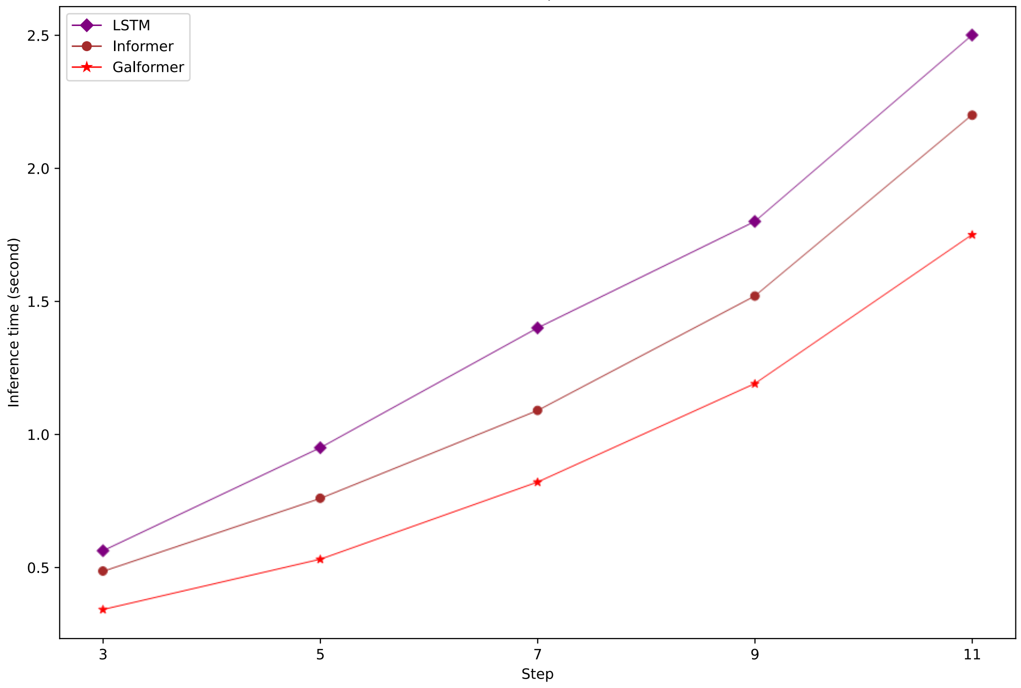


Fig. 9. Testing runtime comparison

4.6. The performance of the Hybrid Loss function

The experiments demonstrate that our proposed hybrid loss function can improve the predictive capability of neural networks for stock market index data to a certain extent, especially in terms of predicting upward and downward trends. Table 3 presents the predictive performance of the hybrid loss function on various models for four stock market indices. Models marked with * indicate that they utilize the proposed hybrid loss function, while the others use the standard MSE loss function.

TABLE 3. Predictive performance of the hybrid loss function

Index	Model	ACC(%)	RMSE	MAPE(%)	MAE	R^2(%)
CSI300	LSTM	50.63	105.23	1.65	83.80	86.63
	LSTM*	50.96	105.10	1.63	83.74	86.65
	Transformer	50.83	65.09	0.99	49.85	94.71
	Transformer*	51.13	63.51	0.95	47.89	95.10
	Informer	51.80	62.37	0.94	47.54	95.19
	Informer*	51.97	62.31	0.94	47.51	95.20
S&P 500	LSTM	50.58	58.63	1.24	45.75	96.40
	LSTM*	50.64	58.47	1.22	43.58	96.42
	Transformer	50.86	39.94	0.82	30.16	98.27
	Transformer*	51.14	38.29	0.80	29.24	98.45
	Informer	51.82	38.12	0.79	29.10	98.53
	Informer*	51.87	38.04	0.78	29.02	98.57
DJI	LSTM	50.62	550.57	1.36	406.54	95.13
	LSTM*	50.71	549.38	1.34	405.22	95.21
	Transformer	50.90	278.59	0.71	214.09	98.78
	Transformer*	51.11	274.46	0.69	206.71	98.83
	Informer	51.79	271.56	0.68	205.89	98.84
	Informer*	51.86	271.53	0.68	205.86	98.85
IXIC	LSTM	50.59	281.56	1.81	225.38	93.80
	LSTM*	50.62	281.47	1.79	224.99	93.91
	Transformer	50.91	175.82	1.04	129.48	97.55
	Transformer*	51.17	170.19	1.02	128.28	97.58
	Informer	51.78	165.80	1.01	126.23	97.83
	Informer*	51.83	165.71	1.00	126.17	97.85

4.7. Comparison of prediction results of different models

In this study, we employed ARIMA, LSTM, Transformer (with MSE loss), Transformer* (with the proposed hybrid loss), Informer and the proposed Galformer to forecast the previously mentioned four datasets, including CSI 300, S&P 500, DJI, and IXIC. The initial 30 days' prediction results for the test set are depicted in Fig. 10. Furthermore, we conducted a thorough comparative analysis of the predictive capabilities of these six models using the previously introduced evaluation indicators. The comprehensive evaluation results for the entire test set are outlined in Table 4.

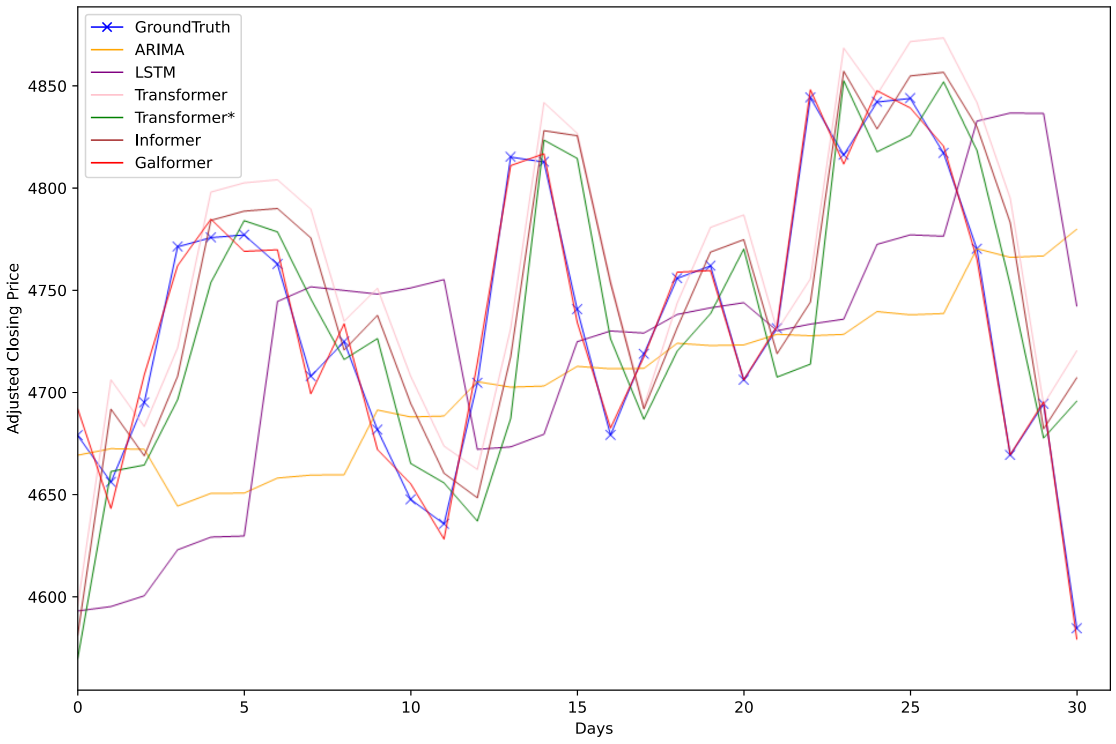
In Fig. 10, the blue curves marked with asterisks represent the true value sequences of the test set, while other colored curves represent the prediction sequences of six comparative models for the test set. It can be observed that the prediction curve of ARIMA deviates the most from the true value curve, followed by LSTM. This indicates that the predictive performance of the relatively simple models ARIMA and LSTM is inferior to that of several more complex neural network prediction models.Furthermore, from the four comparison plots of prediction results, it can be seen that the red curve, representing the predictions of the proposed Galformer model, closely follows the true value sequence, with the trends in both directions showing close alignment. This demonstrates that the Galformer model exhibits higher predictive accuracy in both numerical values and trends compared to the other comparative models.

In Table 4, models marked with '*' used a hybrid loss function, while other models employed an

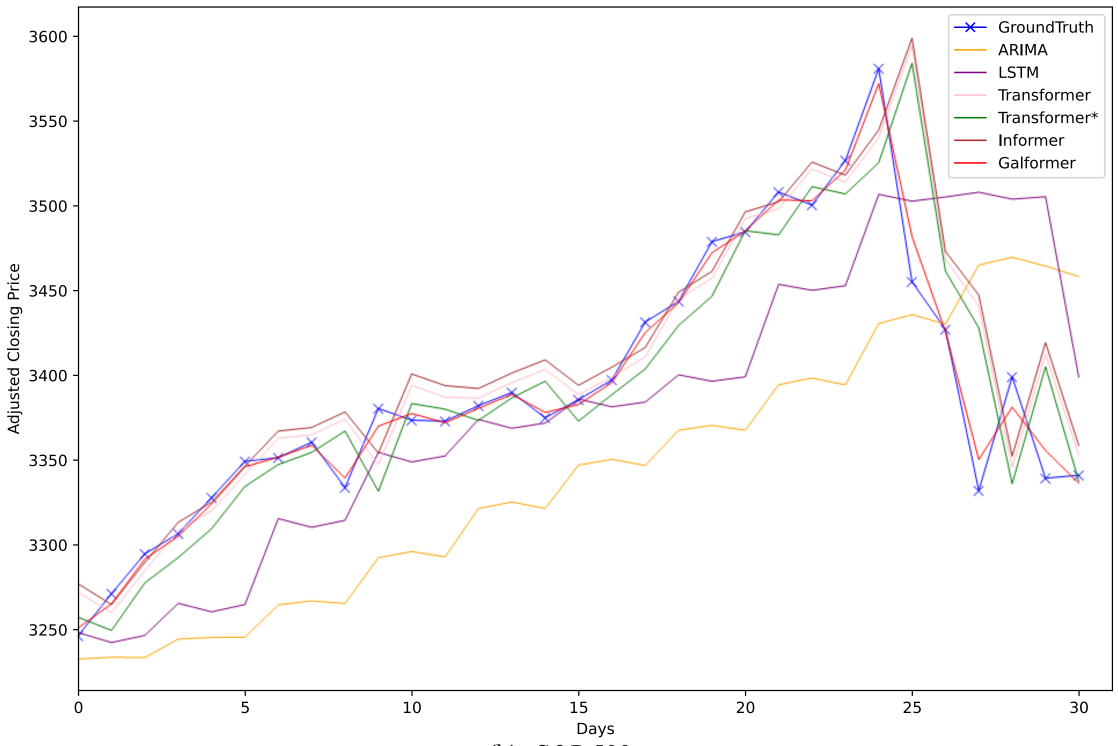
MSE loss function. Underlined entries in Table 4 indicate the highest prediction accuracy for the same stock dataset across models. It can be observed that across various metrics, the prediction accuracy rankings for the four stock indices are as follows:

Galformer > Informer > Transformer* > Transformer > LSTM > ARIMA.

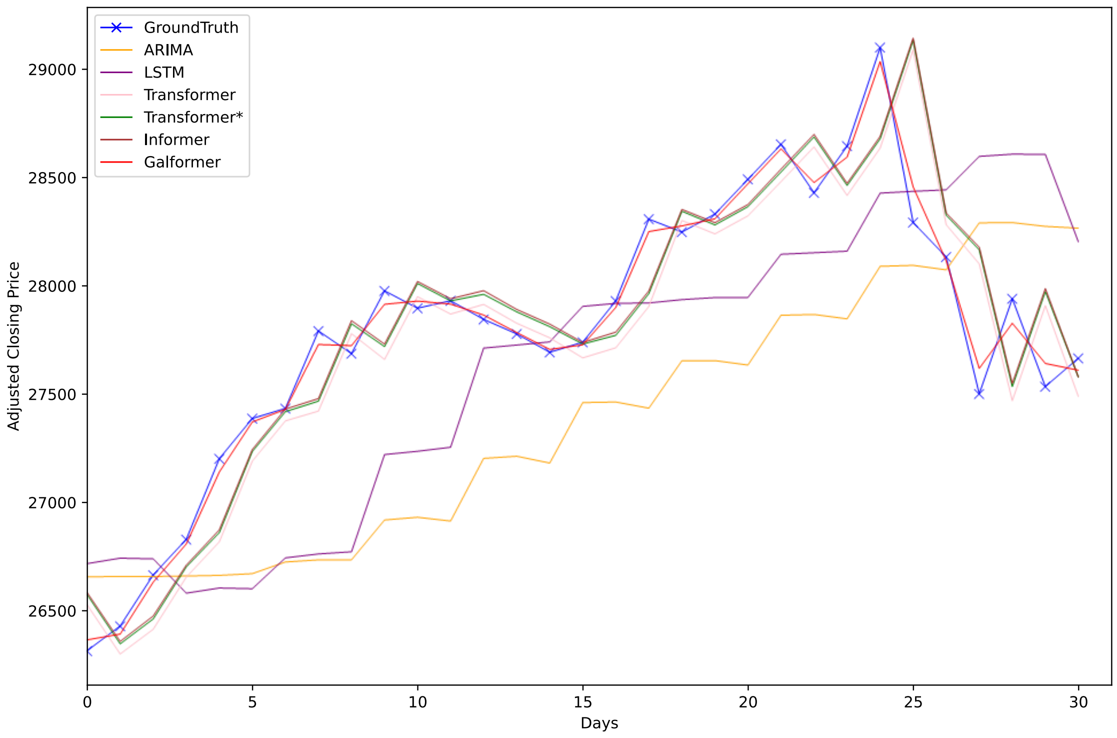
This highlights the effectiveness of the hybrid loss function in enhancing the prediction accuracy of the single Transformer model. Overall, Galformer demonstrates superior prediction accuracy in both numerical values and trends compared to the other comparative models.



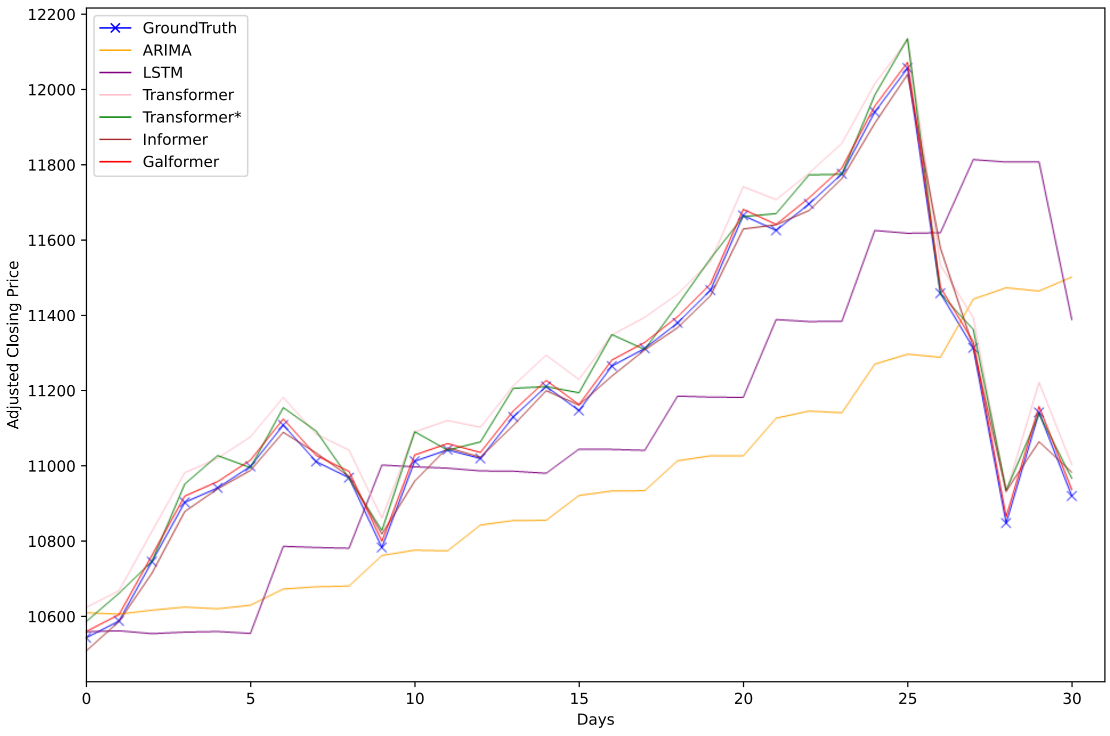
(a) CSI300



(b) S&P 500



(c) DJI



(d) IXIC

Fig. 10. Prediction results of six models

TABLE 4. Numerical representation of predictive performance

Index	Model	ACC(%)	RMSE	MAPE(%)	MAE	R^2(%)
CSI300	ARIMA	50.42	134.09	1.98	101.13	76.41
	LSTM	50.63	105.23	1.65	83.80	86.63
	Transformer	50.83	65.09	0.99	49.85	94.71
	Transformer*	51.13	63.51	0.95	47.89	95.10
	Informer	51.80	62.37	0.94	47.54	95.19
	Galformer	<u>52.75</u>	<u>62.14</u>	<u>0.93</u>	<u>47.15</u>	<u>95.23</u>
S&P 500	ARIMA	50.39	78.76	1.77	65.52	93.53
	LSTM	50.58	58.63	1.24	45.75	96.42
	Transformer	50.86	39.94	0.82	30.16	98.27
	Transformer*	51.14	38.29	0.80	29.24	98.45
	Informer	51.82	38.12	0.79	29.10	98.53
	Galformer	<u>52.69</u>	<u>37.88</u>	<u>0.78</u>	<u>28.93</u>	<u>98.56</u>
DJI	ARIMA	50.44	647.73	1.69	507.13	93.40
	LSTM	50.62	550.57	1.36	406.54	95.13
	Transformer	50.90	278.59	0.71	214.09	98.78
	Transformer*	51.11	274.46	0.69	206.71	98.83
	Informer	51.79	271.56	0.68	205.89	98.84
	Galformer	<u>52.73</u>	<u>271.45</u>	<u>0.67</u>	<u>205.77</u>	<u>98.86</u>
IXIC	ARIMA	50.41	377.78	2.62	327.36	88.80
	LSTM	50.59	281.56	1.81	225.38	93.80
	Transformer	50.91	175.82	1.04	129.48	97.55
	Transformer*	51.17	170.19	1.02	128.28	97.58
	Informer	51.78	165.80	1.01	126.23	97.83
	Galformer	<u>52.56</u>	<u>165.25</u>	<u>0.99</u>	<u>123.66</u>	<u>97.94</u>

4.8. Ablation Study

We perform multi-step predictions on CSI300 and S&P 500, incorporation an ablation study to investigate the contributions of the proposed generative decoder and hybrid loss function.

From Table 5, the following observations can be made:

(1) The underscored data indicates that the generative decoder can improve the prediction accuracy of the model in terms of numerical size to a large extent, which is reflected in the decrease amplitude of RMSE, MAPE, MAE and R^2.

(2) The bolded data signifies that the hybrid loss function can greatly improve the accuracy of the model in trend prediction, evident in the increased ACC.

(3) With the support of generative decoder and hybrid loss function modules, the prediction results of the overall Galformer model have been greatly improved in various indicators compared with Transformer model. Even as the prediction step increases, the prediction accuracy of Galformer does not decrease significantly. This can be attributed to the one-step generative decoder, which mitigates error accumulation associated with iterative predictions.

From Fig. 11, it is evident that: the proposed hybrid loss function exhibits a significant improvement in ACC, while the proposed generative decoder shows notable enhancements in the other four indicators.

TABLE 5. Ablation study of the generative decoder and hybrid loss function, "**G**" represents the proposed generative decoder, while "**L**" denotes the proposed hybrid loss function, "**step**" indicates the number of days for multi-step forecasting.

Index	**Model**	**ACC(%)**	**RMSE**	**MAPE(%)**	**MAE**	**R^2(%)**
CSI300 (3-**step**)	Transformer	50.83	65.09	0.99	49.85	94.71
	Transformer + **G**	51.02	62.51	0.94	47.53	95.04
	Transformer + **L**	**52.39**	64.97	0.95	48.21	95.19
	Galformer	52.75	62.14	0.93	47.15	95.23
CSI300 (5-**step**)	Transformer	50.61	69.42	1.14	53.75	94.06
	Transformer + **G**	50.73	66.31	1.08	51.24	94.75
	Transformer + **L**	**51.96**	68.07	1.10	52.19	94.21
	Galformer	52.33	65.25	1.02	50.12	94.90
CSI300 (7-**step**)	Transformer	50.39	72.95	1.21	56.41	93.49
	Transformer + **G**	51.62	71.84	1.17	55.96	93.72
	Transformer + **L**	**50.71**	69.26	1.12	52.95	94.06
	Galformer	51.97	68.43	1.10	52.57	94.38
S&P 500 (3-**step**)	Transformer	50.86	39.94	0.82	30.16	98.27
	Transformer + **G**	51.05	38.12	0.79	29.24	98.45
	Transformer + **L**	**52.43**	39.57	0.81	29.98	98.34
	Galformer	52.69	37.88	0.78	28.93	98.56
S&P 500 (5-**step**)	Transformer	50.67	41.69	0.89	31.85	97.95
	Transformer + **G**	50.85	39.34	0.83	30.18	98.17
	Transformer + **L**	**52.33**	41.16	0.86	31.47	98.02
	Galformer	52.42	38.91	0.80	29.89	98.26
S&P 500 (7-**step**)	Transformer	50.54	42.73	0.95	32.94	97.58
	Transformer + **G**	50.67	39.98	0.87	30.82	97.79
	Transformer + **L**	**51.95**	42.11	0.91	32.36	97.65
	Galformer	52.13	39.54	0.84	30.57	97.94

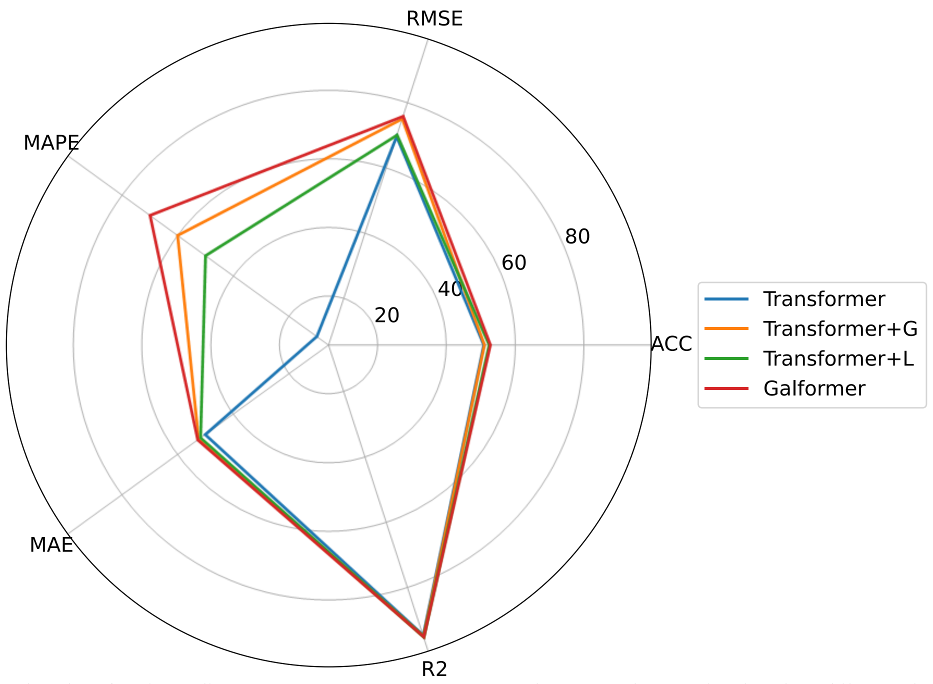


Fig. 11. Radar Chart for Five Indicators Improvement. "**G**" represents the proposed generative decoder, while "**L**" denotes the proposed hybrid loss function. The five indicators are scaled to a range of 0 to 100 for comparison. The farther the vertices of the pentagon are from the center, the higher the accuracy of the predictions.

5. Conclusion

In summary, this paper introduces an innovative transformer-based model with generative decoding and a hybrid loss function for multi-step forecasting of Stock Market Index data. The generative decoding method is specifically proposed to enhance computational speed during the testing phase, while a hybrid trend loss function is introduced to improve trend accuracy in the prediction results.

The comparative experiments demonstrate that the generative decoding method significantly improves computational speed during the testing phase, and the hybrid trend loss function enhances the neural network's trend accuracy in predicting stock indices. The performance of the proposed Galformer model was evaluated on four representative stock indices (CSI 300, S&P 500, DJI, and IXIC) using five evaluation metrics (ACC, RMSE, MAPE, MAE, and R^2), comparing it to four baselines (ARIMA, LSTM, Transformer, and Informer). Comparative studies reveal that the proposed Galformer outperforms the other four models, and subsequent experiments further confirm the feasibility of Galformer for multi-step forecasting.

The Galformer model shows potential for real-world applications in financial decision-making and stock market forecasting by enhancing the accuracy of stock price predictions, optimizing portfolio management, and improving risk assessment strategies. Financial analysts and investors can leverage Galformer's high accuracy in trend prediction to formulate trading decisions based on the projected future trend curves provided by the model.

Despite the excellent performance of Galformer in multi-step stock prediction, it still faces several challenges. The dataset used in the study does not include high-frequency stock data, hence Galformer may not be suitable for high-frequency financial time series prediction. Additionally, the forecasting horizon of Galformer is relatively short, which makes it difficult to conduct long-term stock market predictions. Future research could enhance predictive capabilities by integrating various types of neural network models, leveraging their complementary strengths, or by considering the correlations and influences among different stock indices to design cross-market neural network models. Furthermore, the predictive capability of Galformer could be further enhanced by refining the mixed loss function, for instance, by incorporating an adaptive coefficient to balance trend accuracy effectively. Moreover, the performance of Galformer can be enhanced further by incorporating techniques such as adaptive positional encoding and timestamp encoding.

The Galformer model advances the current state-of-the-art in stock market prediction through its efficient multi-step forecasting capability, surpassing traditional methods in accuracy and computational efficiency. The Galformer model demonstrates superior performance in multi-step stock market prediction compared to traditional methods. However, this study also has limitations. One major limitation is the reliance on historical data that may not fully capture sudden market shifts or external shocks, limiting its real-time applicability. Moreover, the model's performance might vary across different financial markets or during periods of market volatility, which could affect its generalizability. Addressing the identified limitations and conducting further validation across diverse market conditions will be crucial to solidify its practical utility and broader adoption in financial decision-making.

6. Acknowledgments

Funding: This work was supported by the Postgraduate Research & Practice Innovation Program of Jiangsu Province [SJCX23_2088], the Natural Science Foundation of Anhui Province [2008085MF186], and the Collaborative Innovation Program of Universities in Anhui Province [GXXT-2021-028].

7. Data availability

The data that support the findings of this study are available from the authors; however, restrictions apply to the availability of these data, so they are not publicly available. Interested researchers (who meet the criteria for access to confidential data) may contact the corresponding author of this paper for access to the datasets generated or analyzed during the current study.

8. References

[1] Anagnostidis, P., Varsakelis, C., & Emmanouilides, C. J. (2016). Has the 2008 financialcrisis affected stock market efficiency? The case of eurozone. Physica A: Statistical Mechanics and its Applications, 447, 116–128

[2] Murphy, J. J. (1999). Technical analysis of the financial markets: A comprehensive guide to trading methods and applications. Penguin.

[3] Tsai, C.-F., & Hsiao, Y.-C. (2010). Combining multiple feature selection methods for stock prediction: Union, intersection, and multi-intersection approaches. Decision Support Systems, 50, 258–269.

[4] Cavalcante, R. C., Brasileiro, R. C., Souza, V. L., Nobrega, J. P., & Oliveira, A. L. (2016). Computational intelligence and financial markets: A survey and future directions. Expert Systems with Applications, 55, 194–211.

[5] Di Persio, L., & Honchar, O. (2016). Artificial neural networks architectures for stock price prediction: Comparisons and applications. International Journal of Circuits, Systems and Signal Processing, 10, 403–413.

[6] Selvin, S., Vinayakumar, R., Gopalakrishnan, E., Menon, V. K., & Soman, K. (2017). Stock price prediction using LSTM, RNN and CNN-sliding window model. In 2017 International conference on advances in computing, communications and informatics (pp. 1643–1647). IEEE.

[7] Bao, W., Yue, J., & Rao, Y. (2017). A deep learning framework for financial time series using stacked autoencoders and long-short term memory. PLoS One, 12, Article e0180944.

[8] Zhang, K., Zhong, G., Dong, J., Wang, S., & Wang, Y. (2019). Stock market prediction based on generative adversarial network. Procedia Computer Science, 147, 400–406.

[9] Li, Y., Ni, P., & Chang, V. (2020). Application of deep reinforcement learning in stock trading strategies and stock forecasting. Computing, 102, 1305–1322.

[10] Jiang, W. (2021). Applications of deep learning in stock market prediction: recent progress. Expert Systems with Applications, Article 115537.

[11] Xi, E., Bing, S., & Jin, Y. (2017). Capsule network performance on complex data. arXiv preprint arXiv:1712.03480.

[12] Huang, Y., Shen, L., & Liu, H. (2019). Grey relational analysis, principal component analysis and forecasting of carbon emissions based on long short-term memory in China. Journal of Cleaner Production, 209, 415–423.

[13] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., et al. (2017). Attention is all you need. In Advances in neural information processing systems (pp. 5998–6008).

[14] Parmar, N., Vaswani, A., Uszkoreit, J., Kaiser, L., Shazeer, N., Ku, A., et al. (2018). Image transformer. In International conference on machine learning (pp. 4055–4064). PMLR.

[15] Li, N., Liu, S., Liu, Y., Zhao, S., & Liu, M. (2019). Neural speech synthesis with transformer network. In Proceedings of the AAAI conference on artificial intelligence (pp. 6706–6713).

[16] Tetko, I. V., Karpov, P., Van Deursen, R., & Godin, G. (2020). State-of-the-art augmented NLP transformer models for direct and single-step retrosynthesis. Nature Communications, 11, 1–11.

[17] Köksal, A., & Özgür, A. (2021). Twitter dataset and evaluation of transformers for turkish sentiment analysis. In 2021 29th Signal processing and communications applications conference (pp. 1–4). IEEE.

[18] Haoyi Zhou, Shanghang Zhang, Jieqi Peng, Shuai Zhang, Jianxin Li, Hui Xiong, and Wancai Zhang. (2021). Informer: Beyond efficient transformer for long sequence time-series forecasting. In AAAI, 2021.

[19] Haixu Wu, Jiehui Xu, Jianmin Wang and Mingsheng Long. (2021). Autoformer: Decomposition Transformers with Auto-Correlation for Long-Term Series Forecasting. In NeurIPS, 2021.

[20] Lund, R. (2007). Time series analysis and its applications: With r examples. Journal of the American Statistical Association, 102(479), 1079. http://dx.doi.org/10.1198/ jasa.2007.s209.

[21] Long, W., Lu, Z., & Cui, L. (2019). Deep learning-based feature engineering for stock price movement prediction. Knowledge-Based Systems, 164, 163–173.

[22] Luo, Y., & Ji, Y. (2022). Prediction of the Stock Adjusted Closing Price Based On Improved PSO-LSTM Neural Network. 2022 International Conference on Machine Learning and Cybernetics (ICMLC), 43-48.

[23] Shiyang Li, Xiaoyong Jin, Yao Xuan, Xiyou Zhou, Wenhu Chen, Y u-Xiang Wang, and Xifeng Yan. (2019). Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. In NeurIPS, 2019.

[24] Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). Deep Learning. MIT Press.

[25] Lim, B., & Zohren, S. (2020). Time Series Forecasting with Deep Learning: A Survey. 10.1098/rsta.2020.0209.

[26] Y. Ji, A. W. -C. Liew and L. Yang. (2021). "A Novel Improved Particle Swarm Optimization with Long-Short Term Memory Hybrid Model for Stock Indices Forecast," in IEEE Access, vol. 9, pp. 23660-23671, 2021, doi: 10.1109/ACCESS.2021.3056713.

[27] Cho, K.; van Merrienboer, B.; Bahdanau, D.; and Bengio, Y. (2014). On the Properties of Neural Machine Translation: Encoder-Decoder Approaches. In Proceedings of SSST@EMNLP 2014, 103–111.

[28] Sutskever, I.; Vinyals, O.; and Le, Q. V. (2014). Sequence to sequence learning with neural networks.

In NIPS, 3104–3112.

[29] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781.

[30] Williams, R.J., & Zipser, D. (1989). A Learning Algorithm for Continually Running Fully Recurrent Neural Networks. Neural Computation, 1270-1280.