WEP Warwick Evans Publishing

Transactions on Economics, Business and Management Research
ISSN: 2960-1789, eISSN: 2960-2254 | Volume 16
EMFT 2025

Forecasting Realized Volatility of Crude Oil: Can MLP-Based Models, CNN-Based Models and Transformer-Based Models Help?

Guoyu Gu *

School of Economics and Management, Nanjing University of Science and Technology, Nanjing, China

* Corresponding Author Email: guguoyu001017@outlook.com

Abstract. The Shanghai crude oil futures market exudes a distinct speculative attribute, which highlights the importance of Volatility Prediction. Therefore, this study uses the improved models based on MLP, CNN and Transformer for time series forecasting to predict the realized volatility of Shanghai crude oil futures. The results of this study show that MLP-based models, CNN-based models and Transformer-based models are better than the original model in three aspects of extending window, rolling window and long series prediction.

Keywords: Volatility forecasting; Shanghai crude oil futures; Deep learning.

1. Introduction

As an important energy resource and even a financial investment tool, crude oil plays a vital role in modern industrial and economic development. Fluctuations in crude oil prices have a very important impact on the global economy, financial markets and regional stability [1–3]. Therefore, the government, entity enterprises and financial investors have great interest in predicting the price of crude oil [4]. At the same time, there are many factors that can affect the price of crude oil. In addition to the fundamentals of supply and demand, the price of crude oil will also be affected by many complex non-fundamental factors, such as speculation, geopolitical events and economic policy uncertainty [5–7], which need in-depth analysis.

After the escalation of the conflict between Russia and Ukraine in February 2022, the global economy, especially the European economy, is suffering a new round of turbulence after the new crown epidemic [8], which has led to a large-scale economic shock and the rupture of the global supply chain [9]. Fluctuations in energy prices will lead to systemic risk spillovers affecting third-party countries [10], and the conflict between Russia and Ukraine will have a greater spillover effect on the global economy, because Russia and Ukraine are large exporters of key energy [11]. It can be seen that energy plays an important role in the global economic environment, and the change of energy supply will even have a huge spillover effect on countries around the world. If we can predict the price of crude oil, an important part of energy, through certain methods, we can have a better understanding of energy volatility, which can further contribute to the strategic analysis and decision-making of countries or enterprises. At the same time, if the model can accurately predict, it can provide guidance for decision makers. According to the prediction results, decision makers can timely analyze market trends and identify potential risks, which gives them enough time to deal with market risks, prevent major risks, and ensure the stable operation of the economy.

From the perspective of time series prediction, neural network model shows great potential. Among them, transformer [12] has achieved success in natural language processing and computer vision [13,14]. With the success of these aspects, people also pay more and more attention to the role of transformer model in time series prediction. However, the direct application of transformer model to time series prediction has problems such as high complexity and may ignore information. Different scholars have modified transformer to adapt to time series prediction tasks. For example, Zhou et al. [15] proposed an informer model based on probspare self-attention mechanism, and Wu et al. [16] applied the operation of sequence decomposition to transformer model and proposed the autoformer

model. This series of Transformer-based models have achieved good prediction results on data sets such as power, traffic and weather.

However, the Transformer-based model applied to time series prediction has also been questioned. Zeng et al. [17] raised the question whether the Transformer-based model can be applied to time series prediction, and the neural networks dliner and nliner, which are mainly linear layers, can beat the Transformer-based model modified for time series prediction on the relevant data sets. Of course, Liu et al. [18] reflected on the framework of transformer model based on this problem, and proposed a new itransformer model and the corresponding converted method that can be applied to Transformer-based model. At the same time, due to the good performance of dliner and nliner for time series prediction proposed by Zeng et al. [17] and the renewed emphasis on MLP in the academic community, the application of MLP architecture to time series prediction has also been further realized. Chen et al. [19] studied the ability of linear models in time series prediction, proposed time series mixer (TsMixer), and Wang et al. [20] further designed Timemixer model based on the shortcomings of existing models, using multi-scale hybrid architecture, which also showed strong prediction ability in relevant data sets. Therefore, this paper hopes to further explore the prediction ability of neural network in finance, apply MLP based model, CNN-based model, RNN-based model and Transformer-based model to predict the realized volatility of crude oil, and evaluate its prediction ability on the realized volatility of crude oil.

2. Empirical Methods

2.1. MLP-based Models

2.1.1. Multi-Layer Perceptron (MLP)

The multi-layer perceptron is arguably the simplest deep neural network. It consists of multiple layers of neurons, where each layer is connected to the layer above it to receive input and to the layer below it to influence the neurons of the current layer. The way to make the multi-layer perceptron break through the limitations of linear models is to add one or more hidden layers to the network. At the same time, to tap the potential of the multi-layer architecture, we also need an additional key element, which is to apply a non-linear activation function to each hidden unit after the affine transformation. With the activation function, the multi-layer perceptron will not degenerate into a linear model. Since this paper studies a regression problem, the activation function mainly adopts the most popular ReLU function, i.e., Rectified Linear Unit. It is simple to implement and performs well in various prediction tasks, with the formula $ReLU(x) = \max(x, 0)$.

2.1.2. TSMixer

Zeng et al. [17] raised doubts about whether Transformer-based models can be applied to time series forecasting, and proposed neural networks dominated by linear layers, DLinear and NLinear, which can outperform Transformer-based models modified for time series forecasting on relevant datasets. Inspired by this, Chen et al. [19] studied the capabilities of linear models in time series forecasting and proposed Time-Series Mixer (TSMixer), a new architecture designed by stacking multi-layer perceptrons. TSMixer can perform mixing operations along the time and feature dimensions to effectively extract information and achieve better results on relevant datasets. TSMixer alternately uses MLP layers in the time and feature dimensions, which corresponds to time-mixing and feature-mixing, enabling effective capture of temporal patterns and cross-variable information. Meanwhile, TSMixer also adopts the residual design proposed in ResNet, which ensures that TSMixer not only retains the capability of temporal linear models but also continues to utilize cross-variable information.

Specifically, to better utilize cross-variable information, TSMixer uses the alternate use of MLP in the time domain and feature domain. The time-domain MLP is shared among all features, while the feature-domain MLP is shared among all time steps. The interleaved design between the two operations effectively utilizes temporal dependencies and cross-variable information. At the same

time, the complexity and model size of TSMixer are effectively constrained, that is, its full MLP layer design keeps the complexity of TSMixer at O(L+C), which makes it not have the high model complexity and long training time like Transformer-based models. In terms of the specific architecture of TSMixer, it mainly consists of the following five parts: first, Time-mixing MLP, which models temporal patterns in the time series and consists of a fully connected layer, an activation function, and a dropout layer. Second, Feature-mixing MLP, which can fully utilize covariate information and uses a two-layer MLP similar to Transformer to learn complex feature transformations. Third, Temporal Projection, a fully connected layer applied to the time domain, which can not only learn temporal patterns but also map the time series from the original input length L to the target prediction length T. Fourth, Residual Connections. Fifth, Normalization. Due to the existence of time mixing and feature mixing operations, TSMixer applies 2D normalization in both time and feature dimensions.

2.1.3. TimeMixer

For existing models, although the decomposition methods can decompose complex time series into more predictable components (such as seasonality and trends), they usually ignore different patterns of time series at different sampling scales. Among them, periodic analysis methods can decompose time series into multiple components with different cycle lengths, but they usually cannot fully utilize information at different scales to obtain more accurate predictions. Therefore, Wang et al. [20] designed the TimeMixer model using a multi-scale hybrid architecture, which adopts unique Past Decomposable Mixing (PDM) and Future Multipredictor Mixing (FMM). The PDM module mixes the decomposed seasonal and trend components in fine-grained and coarse-grained directions to aggregate micro seasonal and macro trend information. The FMM module combines multiple predictors to utilize the complementary predictive capabilities at different scales to obtain more accurate future predictions.

From the perspective of the specific architecture of the TimeMixer model, one is the Past Decomposable Mixing module, whose purpose is to extract key information from past observations and perform time series modeling through multi-scale representation learning. Firstly, the input sequence is downsampled through average pooling to generate M sequences of different scales, from fine-grained to coarse-grained. Then, feature extraction is performed, and the sequence of each scale is projected into a deep feature space using an embedding layer. Then, seasonal and trend decomposition can be performed, and the sequence of each scale is decomposed into a seasonal part and a trend part using the decomposition method proposed in Autoformer. Seasonal mixing can adopt a bottom-up approach, passing the seasonal information of fine-grained sequences upward to supplement the seasonal modeling of coarse-grained sequences. Finally, trend mixing adopts a top-down approach, passing the trend information of coarse-grained sequences downward to guide the trend modeling of fine-grained sequences. These operations enable the PDM module to generate multi-scale representations containing seasonal and trend information, providing a basis for future predictions. The other is the Future Multipredictor Mixing module, which can utilize the multi-scale information extracted by the PDM module and combine the complementary predictive capabilities of multiple predictors to obtain more accurate future predictions. Specifically, first, multi-scale prediction is performed, and a single linear layer is used to predict the multi-scale representation of each scale to obtain the future prediction result of each scale. Then, the prediction results of all scales are mixed to obtain the final prediction result. Thus, the FMM module can generate the final prediction result for time series forecasting.

2.2. Transformer-based Models

2.2.1. Transformer

The Transformer model is entirely based on the attention mechanism and does not have any convolutional layers or recurrent neural network layers [12]. The Transformer model was initially applied to sequence-to-sequence learning based on text data and later extended to many aspects of

deep learning, such as speech recognition and images. This paper hopes to apply it to time series data such as crude oil prices concerned in this paper. In terms of the architecture of the Transformer model, first, it is necessary to introduce its most important self-attention mechanism. Given an input sequence $x_1, \ldots, x_n$, the self-attention output of the sequence is a sequence of the same length $y_1, \ldots, y_n$. Popularly speaking, for an input x_i, it serves as a query, and all inputs including itself serve as keys. The similarity is calculated by pairing the query with each key to obtain the corresponding weights. All inputs also undergo certain calculations to obtain values. The corresponding weights are multiplied by the corresponding values and summed up to obtain the corresponding output y_i. Performing this operation for each input will yield all outputs. Unlike recurrent neural networks that process inputs iteratively one by one, the self-attention mechanism of the Transformer model abandons sequential operations for parallel computing. To use the sequential information of the sequence, absolute or relative positional information can be injected by adding positional encoding to the input representation. This paper uses the fixed positional encoding based on sine and cosine functions used by the original Transformer model, that is, the corresponding positional encoding is added to the input:

$$p_{i,2j} = \sin\left(\frac{i}{10000^{\frac{2j}{d}}}\right), p_{i,2j+1} = \cos\left(\frac{i}{10000^{\frac{2j}{d}}}\right) \tag{1}$$

In terms of the specific architecture of the Transformer model, from a macro perspective, the encoder of the Transformer is formed by stacking multiple identical layers, each layer having two sub-layers: the first sub-layer is multi-head self-attention aggregation, and the second sub-layer is a position-wise feed-forward network. Meanwhile, inspired by the residual network (Resnet), each sub-layer adopts a residual connection method. The decoder of the Transformer is also formed by stacking multiple identical layers, and the layers use residual connections and layer normalization. In addition to the two sub-layers described in the encoder, the decoder also inserts a third sub-layer between these two sub-layers, namely the encoder-decoder attention layer. In this layer, the query comes from the output of the previous decoder layer, while the keys and values come from the output of the entire encoder. At the same time, different from the multi-head self-attention mechanism of the encoder, each position in the decoder's self-attention can only consider all positions before that position. This masked attention retains the autoregressive property to ensure that the prediction only depends on the generated variables.

2.2.2. Informer

As a brand-new architecture, the Transformer model can outperform other deep learning algorithms in handling many problems. It was initially proposed for machine translation and later extended to natural language processing and image recognition, achieving excellent results. However, directly applying the Transformer model to time series data may not yield better results, that is, when using the Transformer model to process time series data, the following problems may be encountered: (1) The quadratic computation of the self-attention mechanism. The canonical dot-product mechanism of Transformer leads to the time complexity and memory usage of each layer reaching $\mathcal{O}(L^2)$; (2) The input drop when predicting long outputs. The dynamic decoding of Transformer makes distribution inference as slow as RNN models [15]. Therefore, Zhou et al. [15] modified the Transformer model to obtain the Informer model. The most important modification is the design of the ProbSparse self-attention mechanism. Under this mechanism, the model only pairs keys with dominant and important queries, that is,

$$\mathcal{A}(Q, K, V) = Softmax\left(\frac{\bar{Q}K^T}{\sqrt{d}}\right)V \tag{2}$$

Where $\bar{Q}$ represents a sparse matrix that only contains the optimal queries under the sparse metric $M(q,K)$, and the number of optimal queries u is controlled by a constant sampling factor c as $u = clnL_Q$. After using this method, the memory usage of the ProbSparse self-attention mechanism is reduced to $\mathcal{O}(L_K lnL_Q)$. However, traversing all queries to measure $M(q_i,K)$ requires calculating each dot product pair. In addition, the LSE operation has potential numerical stability issues. To solve these problems, the Informer model proposes an empirical approximation method to effectively obtain the query sparse metric, namely the maximum mean measure

$$\bar{M}(q_i,K) = \max_j \left\{\frac{q_i k_j^T}{\sqrt{d}}\right\} - \frac{1}{L_K}\sum_{j=1}^{L_K}\frac{q_i k_j^T}{\sqrt{d}} \tag{3}$$

Under the long-tail distribution, only $U = L_K lnL_Q$ dot product pairs need to be randomly sampled to calculate $\bar{M}(q_i,K)$, while other pairs are padded with zeros, and then the sparse Top-u are selected as $\bar{Q}$. The maximum operator in $\bar{M}(q_i,K)$ is not very sensitive to zero values and is numerically stable. Meanwhile, in practice, the input lengths of queries and keys are equivalent in self-attention calculation, which reduces the memory usage of the ProbSparse self-attention mechanism to $\mathcal{O}(LlnL)$.

In terms of the specific implementation of the Informer model. First, it is necessary to mention the most important ProbSparse self-attention mechanism of the Informer model, which is constructed by inheriting nn.Module. It includes the _prob_QK method, which implements a probabilistic query-key matching method, calculates a sparse attention matrix by randomly sampling a part of the keys, and returns the sampled and sorted query-key matrix and the indices of the sampling positions; and the _get_initial_context method, which calculates the initial context vector according to the mask flag. If a mask is used, the context vector is based on the cumulative sum of values; there is also the _update_context method, which updates the context vector using the calculated attention weights; finally, the forward method transposes the queries, keys, and values for subsequent calculations, calculates the sampled attention scores, adds a scaling factor, obtains the initial context vector, updates the context vector using the selected queries, and returns the updated context vector. After constructing the ProbSparse self-attention mechanism, the corresponding attention layer can be built, and the multi-head attention mechanism can be implemented in this part. Then, another focus is the processing of positional information of time series data. Classes such as PositionalEmbedding, TokenEmbedding, and TemporalEmbedding are constructed respectively to process positional information, and finally, the DataEmbedding class is used to realize the summation of the above processing. Then, we can start building the encoder and decoder. Unlike the Transformer model which does not contain convolutional layers and recurrent neural network layers, both the encoder and decoder of the Informer model use the one-dimensional convolutional layer of nn. conv1d, layer normalization of nn.LayerNorm, and random dropout using nn.Dropout. Taking the encoder as an example, for the input data, first, it is processed using the attention layer containing the ProbSparse self-attention mechanism, and the processed data is randomly dropped using nn.Dropout. Meanwhile, drawing on the operation of the residual network (Resnet), the original data is summed with the above processed data. The data obtained in this way can be layer-normalized to obtain data x. Then, the data x is put into the one-dimensional convolutional layer, activated with the relu function, randomly dropped again using nn.Dropout, then put into the one-dimensional convolutional layer again, and randomly dropped using nn.Dropout to obtain data y. Finally, the layer normalization of x+y is returned. With the decoder and encoder, they can be combined to build the final Informer class. For the input feature data, first, the DataEmbedding is used to process its positional information, then the encoding layer is used to process it to obtain enc_out. At the same time, the DataEmbedding is also used to process the positional information of the input label data to obtain dec_out. Then, the decoder can be used to process enc_out and dec_out to obtain out. Finally, the linear layer is used to convert

out to the required length and return it. The training and prediction of the model are basically consistent with the previous convolutional neural networks and recurrent neural networks.

2.2.3. Autoformer

In addition to the Informer model, there are other improved models based on the Transformer model architecture, such as the Autoformer model. Transformer-based time series prediction models capture dependencies between moments through the self-attention mechanism, but in long-term sequence prediction, there are still shortcomings. For example, complex temporal patterns in long sequences make it difficult for the attention mechanism to find reliable temporal dependencies. Transformer-based models have to use sparse forms of attention mechanisms to deal with the problem of quadratic complexity, but this causes a bottleneck in information utilization [16]. Therefore, Haixu Wu et al. [16] proposed the Autoformer model, breaking through the traditional method of using sequence decomposition as preprocessing, and proposed a Decomposition Architecture that can decompose more predictable components from complex temporal patterns. At the same time, based on the theory of stochastic processes, an Auto-Correlation Mechanism is proposed to replace the point-wise connected attention mechanism, realizing sequence-level connection and $\mathcal{O}(LlnL)$ complexity, breaking the bottleneck of information utilization.

In terms of the specific architecture, the Autoformer model also adopts the same encoder-decoder architecture as the Transformer model. The difference is that the Autoformer model creatively uses the Auto-Correlation Mechanism and Series Decomposition Unit (Series Decomp) in the encoder-decoder. The sequence decomposition unit is based on the sliding average idea, smoothing periodic terms and highlighting trend terms. From the formula,

$$\mathcal{X}_t = \text{AvgPool}\big(\text{Padding}(\mathcal{X})\big), \mathcal{X}_s = \mathcal{X} - \mathcal{X}_t \tag{4}$$

Where $\mathcal{X}$ is the latent variable to be decomposed, and $\mathcal{X}_t$ and $\mathcal{X}_s$ are the trend term and periodic term respectively. The auto-correlation mechanism includes Period-based dependencies and Time delay aggregation. Period-based dependencies are based on the theory of stochastic processes. For a real discrete-time process $\{\mathcal{X}_t\}$, we can calculate its autocorrelation coefficient as follows:

$$\mathcal{R}_{\mathcal{X}\mathcal{X}}(\tau) = \lim_{L\to\infty} \frac{1}{L}\sum_{t=1}^{L} \mathcal{X}_t \mathcal{X}_{t-\tau} \tag{5}$$

Time delay aggregation is to realize sequence-level connection and aggregate information of similar subsequences. According to the estimated cycle length, first, the Roll () operation is used for information alignment, and then information aggregation is performed. Here, the form of query, key, and value is still used, so that it can seamlessly replace the self-attention mechanism. The specific formulas are as follows:

$$\begin{aligned} \tau_1, \cdots, \tau_k &= \underset{\tau\in\{1,\cdots,L\}}{\arg \text{Topk}} \left(\mathcal{R}_{\mathcal{Q},\mathcal{K}}(\tau)\right) \\ \hat{\mathcal{R}}_{\mathcal{Q},\mathcal{K}}(\tau_1), \cdots, \hat{\mathcal{R}}_{\mathcal{Q},\mathcal{K}}(\tau_k) &= \text{SoftMax}\left(\mathcal{R}_{\mathcal{Q},\mathcal{K}}(\tau_1), \cdots, \mathcal{R}_{\mathcal{Q},\mathcal{K}}(\tau_k)\right) \\ \text{Auto-Correlation}(\mathcal{Q},\mathcal{K},\mathcal{V}) &= \sum_{i=1}^{k} \text{Roll}(\mathcal{V},\tau_i)\hat{\mathcal{R}}_{\mathcal{Q},\mathcal{K}}(\tau_i) \end{aligned} \tag{6}$$

In terms of the specific implementation of the Autoformer model, first, the input data needs to be encoded, which is basically the same as the Informer model, but the Autoformer model does not need positional encoding due to its particularity. Next, the auto-correlation mechanism module is constructed. The time_delay_agg_training method is designed, which is the auto-correlation mechanism in training mode, used to discover periodic dependencies. It calculates the correlation between queries and keys using Fourier transform and convolution, calculates the top k positions according to the scaling factor and sequence length, calculates the weights using the softmax function, applies time delay aggregation, multiplies the values by the weights and delay indices, and finally returns the aggregated values. The time_delay_agg_inference method is designed, which is basically the same as the previous step but is used for the auto-correlation mechanism in inference mode for time delay aggregation. Then, the time_delay_agg_full method is designed for integration, which is the complete auto-correlation mechanism, including periodic dependency discovery and time delay aggregation. Finally, the forward method executes the auto-correlation mechanism, adjusts the keys and values according to the sequence length, calculates the correlation between queries and keys using Fourier transform and convolution, and selects the time_delay_agg_training or time_delay_agg_inference method for time delay aggregation according to the training mode, and returns the aggregated values. Another thing to design is the sequence decomposition unit. First, the moving_avg class is designed. In this class, two parameters are defined, including the moving average window size (kernel_size) and stride (stride). During initialization, the above parameters are set, and an nn. AvgPool1d layer is created to calculate the moving average of the time series. In the forward method, zero padding is added at the beginning and end of the time series to avoid data loss when the window slides, the moving average is calculated using the nn. AvgPool1d layer, and finally the processed time series is returned. Then, the series_decomp class is designed. In this step, the moving_avg module is used to calculate the moving average of the time series, and the difference between the time series and the moving average is calculated to obtain the decomposed trend part and seasonal part, which are finally returned. The construction of the encoder and decoder is basically the same as that of the Informer model, except that the Autoformer model adopts the sequence decomposition and auto-correlation mechanism constructed earlier.

2.2.4. Reformer

Kitaev et al. [21] proposed the improved Reformer due to the high training cost of the Transformer model. It uses locality-sensitive hashing attention to replace the original dot-product operation, reducing the model complexity from $O(L^2)$ to $O(L\log L)$, where L is the length of the sequence. In addition, reversible residual layers are used to replace the original standard residual layers, which enables the model to store activations only once during training. Among them, the most important is locality-sensitive hashing attention. When performing the self-attention mechanism, for each query q, the model only pays attention to the 32 or 64 keys k closest to it, which can effectively improve model efficiency. To find the keys closest to the query q among the keys, locality-sensitive hashing can be used, which is achieved by assigning different hash values to different vectors.

2.2.5. ETSformer

Woo et al. [22] modified the Transformer model using the principle of exponential smoothing and proposed ETSformer. Specifically, the model proposed a new exponential smoothing method and frequency attention to replace the self-attention mechanism in the original Transformer model, thereby improving accuracy and efficiency.

ETSformer model induces biases on time series data through layer-wise decomposition, growth, and seasonal decomposition. By utilizing the high capacity of the deep framework and effective residual learning schemes, the ETSformer model can extract a series of potential growth and seasonal patterns and model their complex changes. Secondly, the ETSformer model introduces a new exponential smoothing attention and frequency attention to replace the original ordinary attention. In the exponential smoothing attention mechanism, attention scores are constructed based on the relative time lag of queries, while frequency attention uses Fourier transform to realize the extraction of

seasonal patterns. Finally, the prediction is composed of level, trend, and seasonal components, which makes the model more interpretable.

2.2.6. FEDformer

Zhou et al. [23] proposed the FEDformer model due to the high computational cost of the Transformer model and its inability to capture the global view of time series. FEDformer combines Transformer with seasonal trend decomposition methods, where the decomposition method can capture the global distribution of time series, and the Transformer architecture can capture more detailed structures. At the same time, FEDformer utilizes the fact that most time series often have sparse responses in well-known bases and develops a frequency-enhanced method.

From the perspective of the FEDformer architecture, the innovative architecture of FEDformer is mainly based on the encoder and decoder of Frequency Enhanced Attention. In frequency-enhanced attention, there are mainly two versions: FEB-f (based on Fourier transform) uses discrete Fourier transform to convert time-domain signals into frequency-domain and randomly selects a set of frequency components for feature extraction and representation learning. FEB-w (based on wavelet transform) uses discrete wavelet transform to decompose time-domain signals into sub-bands of different scales and performs FEB-f processing on each sub-band separately to capture local features and periodic patterns in time series. The advantages of this module include the following aspects: first, it can reduce computational complexity. By randomly selecting frequency components, the computational complexity of Transformer is reduced from quadratic to linear, enabling it to effectively process long sequence data. Second, it enhances the learning of global and local features. FEB-f can capture global features in time series, such as trends and periodic patterns, while FEB-w can capture local features in time series, such as spikes and mutations. Third, it improves prediction accuracy. By better capturing the features of time series, this module can improve the prediction accuracy of the model, especially in long-term prediction.

2.2.7. Flowformer

Since existing models reduce the complexity of the Transformer model by designing linear-time attention mechanisms through the combination of similarity decomposition and matrix multiplication, such operations sacrifice the generalization ability of the model. Therefore, Wu et al. [24] proposed the Flowformer model. The Flowformer model uses flow network theory, regards attention as information flow, aggregates from values to results through learned flow capacities, and avoids attention degradation through source competition and sink allocation mechanisms while maintaining model generalization.

From the perspective of the specific architecture of the Flowformer model, the most important is the multi-head Flow-Attention mechanism. First, it accepts queries Q, keys K, and values V from the previous layer or embedding layer. Second, queries and keys are regarded as nodes in the information flow network, values as information sources, and results as information sinks. Then, by introducing flow conservation mechanisms, sources and sinks are constrained respectively to realize competition and allocation mechanisms, thereby avoiding attention degradation. Finally, in the calculation process, by normalizing each sink, competition is generated among sources to obtain competitive source information, and matrix multiplication is used to aggregate the competitive source information to obtain aggregated source information. Then, by normalizing the outflow of each source, allocation weights are obtained, and the aggregated source information is filtered to obtain the final result. In addition to the multi-head Flow-Attention mechanism, the architecture of Flowformer also includes common parts in some Transformer-based models, including feed-forward networks to accept results from the multi-head Flow-Attention mechanism, perform linear transformation and activation function operations, and extract features; layer normalization to normalize the input and output of layers to improve the stability and performance of the model; residual connections to add the input and output of layers to retain information and improve the performance of the model.

2.2.8. Crossformer

Zhang et al. [25] considered that existing Transformer-based models for time series forecasting have the defect of ignoring cross-dimensional dependencies and proposed Crossformer that can use cross-dimensional dependencies for forecasting. From the perspective of the specific architecture of Crossformer, first is Dimension Segment-wise Embedding (DSW), which can segment the sequence of each dimension into segments and embed them into feature vectors, outputting a two-dimensional vector array containing time and dimension information. Second is the Two-Stage Attention (TSA) layer. In the cross-time stage, multi-head self-attention is applied to each dimension to capture cross-time dependencies. In the cross-dimensional stage, a routing mechanism is used to reduce computational complexity while capturing cross-dimensional dependencies. Finally, it is divided into an encoder-decoder structure (HED), where the encoder uses two-stage attention layers and segment merging to capture dependencies of different scales, and the decoder uses features of different scales output by the encoder for prediction. Therefore, the Crossformer model can explicitly utilize cross-dimensional dependencies, more effectively capture the dependencies between different dimensions, thereby improving prediction accuracy; at the same time, the two-stage attention mechanism and hierarchical structure effectively reduce the computational complexity of the model, enabling it to process large-scale datasets.

2.2.9. PatchTST

Due to the high time and space complexity of the Transformer model in time series forecasting, resulting in huge computation and memory consumption, and the Transformer model may ignore the importance of local semantic information in time series and be difficult to capture the connections between elements in the sequence. Therefore, Nie et al. [26] proposed the PatchTST model, which is greatly improved based on the Transformer model. In this model, Patching technology and Channel-independence are proposed, which can effectively reduce time and space complexity.

In terms of the specific architecture of PatchTST, first, for data input processing, multivariate time series data is divided into multiple univariate time series, and each univariate time series is divided into multiple overlapping or non-overlapping subsequence patches according to a fixed step size; second, the encoder of PatchTST, where each input token corresponds to a patch, and Instance Normalization operation is applied to each patch to reduce the distribution difference between training and test data. Through the multi-head attention mechanism and feed-forward network, the dependencies between patches are learned and feature representations are generated; finally, in the prediction stage, the encoded feature representations are flattened and predicted through a linear layer, and the MSE loss function is used to evaluate the difference between the prediction result and the true value.

2.2.10. iTransformer

Since Zeng et al. [17] raised doubts about whether Transformer-based models can be applied to time series forecasting, Liu et al. [18] re-examined the architecture of Transformer and found that traditional Transformer models embed multiple variables at the same time point into the same token, leading to confusion of correlations between multiple variables, difficulty in learning variable-centered representations, and may result in invalid attention maps. At the same time, traditional Transformer uses a permutation-invariant attention mechanism in the time dimension, which may cause problems in time series data, because different time points may represent different meanings, and the permutation-invariant attention mechanism cannot distinguish these differences. Therefore, Liu et al. [18] proposed iTransformer accordingly, which can be regarded as a new model or an idea that can be applied to other Transformer-based models, such as combining this idea with the Informer model to form an inverted-informer model. The components of the iTransformer model itself are effective for multivariate time series forecasting. The problem lies in the way the architecture is used. Therefore, iTransformer flips the time series, independently embeds each time series as a variable token, uses the self-attention mechanism to capture multivariate correlations, and uses the feed-forward network to learn time series representations.

From the perspective of the specific architecture of the iTransformer model, first is variable tokenization, which independently embeds each time series as a variable token, which can more clearly express the correlations between multiple variables and learn variable-centered representations. Second is the flipped architecture, which applies the attention mechanism of traditional Transformer to the variable dimension, which can better capture the correlations between multiple variables and improve the generalization ability of the model. Finally, the feed-forward network learns time series representations, and the feed-forward network can learn more complex representations, thereby improving the prediction accuracy of the model.

2.3. CNN-based Models

2.3.1. Convolutional Neural Network (CNN)

Convolutional neural networks are a class of powerful neural networks designed to process image data. With the development of convolutional neural networks, they can also be applied to one-dimensional sequence structure tasks. The birth of convolutional neural networks is mainly because multi-layer perceptrons require a large number of parameters when processing image data, which makes training such models difficult. Convolution has the advantages of translation invariance and locality.

For the layers in the convolutional neural network, first is the convolutional layer. The convolutional layer performs a cross-correlation operation on the input and the convolution kernel and generates an output after adding a scalar bias. That is to say, the two trained parameters in the convolutional layer are the convolution kernel and the scalar bias. Next is the setting of padding and stride. When applying multiple layers of convolution, we often lose edge pixels. Since we use small convolution kernels, we may only lose a few pixels for any single convolution, but as we apply many consecutive convolutional layers, the accumulated number of lost pixels will increase. A simple way to solve this problem is padding, which fills elements at the edges of the input image. This is a two-dimensional processing method. The situation is similar in one dimension, except that it is filled on both sides of the one-dimensional data instead of four sides of the two-dimensional data. This paper sets padding=1, that is, fills one bit on both sides of the one-dimensional data. When calculating cross-correlation, the convolution window slides over the one-dimensional data. Sometimes, to perform efficient calculation or reduce the number of samples, the convolution window can skip intermediate positions and slide multiple elements each time. In this paper, stride=1 is set, that is, no skipping. Next is the pooling layer. Usually, when processing images, we hope to gradually reduce the spatial resolution of the hidden representation and aggregate information, so that as the number of layers in the neural network increases, each neuron has a larger receptive field (input) that it is sensitive to. The same is true for one-dimensional data. Pooling can reduce the sensitivity of the convolutional layer to position and the sensitivity to spatially downsampled representations. Similar to the convolutional layer, the pooling layer operator consists of a fixed-shaped window that slides over all regions of the input according to its stride size and calculates an output for each position traversed by the fixed-shaped window. However, unlike the cross-correlation operation between the input and the convolution kernel in the convolutional layer, the pooling layer does not contain parameters, and the pooling operation is deterministic. We usually calculate the maximum and average values of all elements in the pooling window, which are called max pooling and average pooling respectively. This paper adopts the form of max pooling layer.

In terms of the specific implementation of CNN, first, we also create our own data reading method by subclassing torch.utils.data.Dataset. In this process, the torch.tensor function is used to convert the read data into a tensor form. Next, the model is built by subclassing nn.Module. The model is mainly divided into two parts: one-dimensional convolution composed of nn. Conv1d, nn.ReLU, and nn.MaxPool1d; and a fully connected layer composed of nn.Linear and nn.ReLU. The result obtained using one-dimensional convolution needs to be flattened into an input acceptable to the fully connected layer using nn.Flatten.

2.3.2. TimesNet

Wu et al. [27] believed that actual time series data often have complex temporal patterns, including multiple periodicities, trends, seasonality, etc. Existing modeling methods based on 1D time series are difficult to capture these complex temporal patterns. Among them, RNN is difficult to capture long-distance dependencies and has low efficiency. The locality of TCN convolution kernels limits the modeling of long-distance dependencies. The attention mechanism of Transformer is difficult to directly find reliable dependencies from scattered time points. Therefore, Wu et al. [27] proposed the TimesNet model that can expand temporal changes into 2D space by utilizing the multi-periodicity of time series.

From the perspective of the specific architecture of TimesNet, the most important is the module called TimesBlock. First is the embedding layer, which converts the original 1D time series input into embedding vectors for subsequent feature extraction; second is Period Discovery, which uses Fourier transform to analyze the embedding vectors, discovers the periodicity in the time series, and obtains the corresponding cycle length by calculating the amplitude of each frequency and selecting the largest k frequencies; then is 2D tensor conversion. Based on the discovered cycle length, the 1D time series is converted into k 2D tensors, where each 2D tensor represents the temporal changes within a specific cycle. The columns and rows of the 2D tensor represent internal periodic changes and external periodic changes respectively; in the Inception Block, feature extraction is performed on each 2D tensor. The Inception Block contains multiple 2D convolution kernels of different scales, which can effectively capture multi-scale temporal changes. The parameter sharing design ensures that the model size does not increase with the increase of the number of cycles; finally, Adaptive Aggregation, which fuses the features extracted from the k 2D tensors, uses the Softmax function to normalize the amplitude corresponding to each cycle as the fusion weight, multiplies the features of each cycle by their weights, and accumulates them to obtain the final output.

TimesNet adopts a modular architecture, and each TimesBlock can independently process temporal changes within a cycle. This design enables the model to flexibly adapt to time series data of different cycle lengths and effectively capture temporal changes brought about by different cycles. TimesNet uses a parameter sharing design, so that the model size does not increase with the increase of the number of cycles, which enables the model to reduce computational complexity and memory usage while maintaining performance. At the same time, the 2D tensor conversion and Inception Block design of TimesNet make the model easier to interpret, and it can be intuitively observed how the model captures temporal changes brought about by different cycles.

2.4. RNN-based Models

2.4.1. Recurrent Neural Network

Unlike convolutional neural networks that can effectively process spatial information, recurrent neural networks (RNN) can handle sequence information well. Recurrent neural networks determine the current output by introducing state variables to store past information and current input. Suppose we have a mini-batch input $x_t \in R^{n\times d}$ at time step t. In other words, for a mini-batch of n sequence samples, each row of x_t corresponds to a sample at time step t from the sequence. Next, $h_t \in R^{n\times h}$ is used to represent the hidden variable at time step t. Different from the multi-layer perceptron, we save the hidden variable h_{t-1} from the previous time step here and introduce a new weight parameter$W_{hh} \in R^{h\times h}$ to describe how to use the hidden variable from the previous time step at the current time step. Specifically, the hidden variable at the current time step is calculated jointly by the input at the current time step and the hidden variable from the previous time step, that is,

$$h_t = \phi(x_t W_{xh} + h_{t-1} W_{hh} + b_n) \tag{7}$$

The specific flowchart is shown in the following figure. These variables capture and retain the historical information of the sequence up to its current time step, just like the state or memory of the neural network at the current time step.

In terms of specific implementation, this paper reads data in a sequential partitioning manner. In this part, a relatively small random number can be set through the random.randint function, and data is read starting from this position and converted into a tensor form. Then, small-batch subsequences are selected according to the mini-batch setting to build the model. Then, the model can be built by subclassing nn.Module. The parameters of the recurrent layer are set through nn.RNN, and the linear layer is set through nn.Linear. The linear layer can convert the data output by the recurrent layer into the required data form and quantity. At the same time, a gradient clipping function can be designed to avoid the problem that the optimization algorithm cannot converge due to excessive gradients. The gradient g can be clipped by projecting it back to a ball of a given radius, that is, $g \leftarrow \min\left(1, \frac{\theta}{\|g\|}\right)g$. By doing this, the gradient norm will never exceed θ, and the direction of the updated gradient is completely consistent with the original direction of g. It also has an additional effect of limiting the impact of any given mini-batch data on the parameter vector, which endows the model with a certain degree of stability. Then, the recurrent neural network can be trained. The training method is basically the same as the previous multi-layer perceptron and convolutional neural network, except that gradient clipping is performed after backpropagation and before parameter update.

2.4.2. Gated Recurrent Unit

A common problem with recurrent neural networks in practice is numerical instability. Although we have adopted techniques such as gradient clipping to alleviate this problem, it is still necessary to design more complex sequence models to further handle it, that is, using the Gated Recurrent Unit (GRU) introduced in this section and the Long Short-Term Memory (LSTM) network introduced in the next section.

The key difference between the gated recurrent unit and the ordinary recurrent neural network is that the former supports gating of the hidden state, which means the model has a dedicated mechanism to determine when to update the hidden state and when to reset the hidden state, and these mechanisms are learnable. First, there are the reset gate and update gate. The reset gate allows the number of past states we "may still want to remember", and the update gate allows us to control how many of the new states are copies of the old states. Mathematically, for a given time step t, assuming the input is a mini-batch $X_t \in \mathbb{R}^{n\times d}$ (number of samples n, number of inputs d), and the hidden state from the previous time step $H_{t-1} \in \mathbb{R}^{n\times h}$ (number of hidden units h), the reset gate $R_t = \sigma(X_t W_{xr} + H_{t-1}W_{hr} + b_r)$ and the update gate $Z_t = \sigma(X_t W_{xz} + H_{t-1}W_{hz} + b_z)$ can be calculated accordingly. At the same time, the activation function here is the sigmoid function, which can convert the input value into the interval $(0,1)$. Next is the candidate hidden state. In this step, the reset gate R_t is integrated with the regular hidden state H_t update mechanism to obtain the candidate hidden state $\widetilde{H}_t = \tanh\left(X_t W_{xh} + (R_t \odot H_{t-1})W_{hh} + b_h\right)$ at time step t. The non-linear activation function tanh used here ensures that the values in the candidate hidden state are within the interval $(-1,1)$. Finally, the hidden state. At this time, the update gate Z_t can be used to determine the extent to which the new hidden state comes from the old hidden state and the new candidate hidden state, that is, $H_t = Z_t \odot H_{t-1} + (1 - Z_t)\odot \widetilde{H}_t$. The above shows that the reset gate of the gated recurrent unit helps capture short-term dependencies in the sequence, and the update gate helps capture long-term dependencies in the sequence. In terms of specific implementation, the implementation of the gated recurrent unit is basically the same as that of the recurrent neural network, except that nn.GRU is used for the gated recurrent unit instead of nn.RNN used for the recurrent neural network when building the model.

2.4.3. Long Short-Term Memory Network

The long short-term memory network is one of the earliest models used to solve the problems of long-term information preservation and short-term input loss in latent variable models. First, there are the

input gate, forget gate, and output gate. The input at the current time step and the hidden state from the previous time step are fed into the gates of the long short-term memory network and processed using the sigmoid function. Specifically, the input gate $I_t = \sigma(X_t W_{xi} + H_{t-1} W_{hi} + b_i)$, the forget gate $F_t = \sigma(X_t W_{xf} + H_{t-1} W_{hf} + b_f)$, and the output gate $O_t = \sigma(X_t W_{xo} + H_{t-1} W_{ho} + b_o)$. Next is the candidate memory cell, using the tanh function as the activation function, $\tilde{C}_t = \tanh(X_t W_{xc} + H_{t-1} W_{hc} + b_c)$. Then, the memory cell. The input gate I_t controls how much new data from $\tilde{C}_t$ is adopted, while the forget gate F_t controls how much content from the past memory cell C_t is retained, that is, $C_t = F_t \odot C_{t-1} + I_t \odot \tilde{C}_t$. Finally, the hidden state. This is where the output gate plays a role: $H_t = O_t \odot \tanh(C_t)$. In terms of specific implementation, the implementation of the long short-term memory network is basically the same as that of the recurrent neural network, except that nn. LSTM is used for the gated recurrent unit instead of nn.RNN used for the recurrent neural network when building the model.

3. Data

3.1. Realized Volatility

Realized volatility is defined as the sum of intraday squared returns, with the formula

$$\mathrm{RV}_t = \sum_{j=1}^{\frac{1}{\delta}} r_{t,j}^2 \tag{8}$$

Where δ is the sampling frequency under high-frequency data, and $r_{t,j}^2$ is the square of the j-th intraday return reported on trading day T.

3.2. Data Description

The dataset used in this paper includes high-frequency data of Shanghai crude oil futures at 5-minute intervals, spanning from March 26, 2018, to August 25, 2021 (excluding holidays), totaling 830 trading days. By recording 1 data point every 5 minutes, a total of 112 high-frequency data records can be generated each day, that is, the realized volatility of the day can be calculated through the 112 high-frequency data of each day, and finally, the time series of realized volatility is obtained, as shown in the following figure. In this study, the expanding window method is adopted, with 600 periods as the sample period, predicting one period each time, and then incorporating the next period's data into the sample period for prediction, and so on to obtain the prediction sequence. The corresponding evaluation criteria are calculated to compare the prediction capabilities of different models.

3.3. Evaluation Criteria

This paper mainly uses a series of indicators to evaluate the results of various prediction models. The specific indicators and calculation methods are as follows. First, MAE (Mean Absolute Error):

$$\mathrm{MAE} = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i| \tag{9}$$

MSE (Mean Squared Error):

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{10}$$

RMSE (Root Mean Squared Error):

$$\mathrm{RMSE} = \sqrt{\mathrm{MSE}} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{11}$$

4. Results and Discussion

4.1. Data Processing and Hyperparameter Setting

In this paper, the original data input is normalized by the following formula $\tilde{x}_{\mathrm{t}} = \frac{x_i - \mu}{\sigma}$ to reduce the inherent volatility of the dataset, where x_i represents the initial input data, $\tilde{x}_{\mathrm{t}}$ represents the normalized data, and μ and σ represent the mean and standard deviation of the original input data x_i respectively. For the sake of not using future data, normalization is only performed on the data within the sample period. At the same time, when denormalizing the predicted values, the mean and standard deviation calculated from the data within the sample period are also used. During the training process of the relevant neural networks used in this paper, the learning rate is set to 0.0001, the optimizer is Adam, and the loss function is MSE. Next is the setting and explanation of the parameters used in each model. Of course, not all the following parameters will be used for a model. seq_len is the input sequence length in each batch, set to 96, that is, using the past 96 periods to predict the next period; pred_len is the prediction sequence length, set to 1, that is, only predicting the volatility of the next period; enc_in is the sequence dimension input to the encoder, set to 1, that is, this paper only uses one time series data of realized volatility and inputs it into the encoder; dec_in is the sequence dimension input to the decoder, set to 1, that is, the data input to the decoder has only one dimension; embed is the timestamp encoding method, and this paper selects timeF; c_out is the output sequence dimension, set to 1, that is, the obtained prediction sequence has only one dimension; n_heads is the number of multi-head attention mechanisms, set to 8; e_layers and d_layers are the number of encoder and decoder layers, and this paper sets e_layers to 2 and d_layers to 1. Other parameters basically follow the default parameters used by the model proposers.

4.2. Out-of-Sample Prediction Results

Table 1 shows the prediction evaluation indicators of TsMixer, TimeMixer, CNN, TimesNet, RNN, GRU, LSTM, Informer, Autoformer, Reformer, ETSformer, FEDformer, Flowformer, Crossformer, PatchTST, Inverted-Transformer, Inverted-Informer, Inverted-Flowformer, and Inverted-Reformer. From the results, in terms of the MAE indicator, the Inverted-Informer model among the Transformer-based models achieved the lowest MAE. Specifically, the MAE of Inverted-Informer achieved an 84% improvement compared to the highest CNN, and a 0.4% improvement compared to the second lowest Crossformer. In terms of MSE and RMSE, the Informer model among the Transformer-based models achieved the lowest MSE and RMSE. Specifically, the MSE of the Informer model achieved a 325% improvement compared to the highest CNN, and the Informer model also achieved a 0.3% improvement compared to the second lowest TimeMixer. It can be seen that compared with those neural networks that have existed for many years, the new models proposed in recent years based on the improvement of traditional architectures have shown more powerful prediction capabilities. However, it is worth noting that this study did not perform parameter tuning, and all models were trained based on consistent parameters. Therefore, there may be cases where the optimal capabilities

of the models cannot be reflected under the default parameters, which is also worthy of further exploration.

Table 1 Out of sample prediction results based on extending window

Category	Model	MAE	MSE	RMSE
MLP-based model	TsMixer	0.00019975	0.00000007	0.00027018
	TimeMixer	0.00020176	0.00000007	0.00026829
CNN-based model	CNN	0.00035495	0.00000031	0.00055271
	TimesNet	0.00022500	0.00000010	0.00031927
RNN-based model	RNN	0.00031255	0.00000026	0.00050975
	GRU	0.00026584	0.00000021	0.00045327
	LSTM	0.00024585	0.00000020	0.00044556
Transformer-based model	Informer	0.00020114	0.00000007	0.00026784
	Autoformer	0.00023976	0.00000010	0.00031038
	Reformer	0.00020067	0.00000007	0.00027137
	ETSformer	0.00029645	0.00000020	0.00044455
	FEDformer	0.00026359	0.00000013	0.00035568
	Flowformer	0.00020939	0.00000009	0.00029825
	Crossformer	0.00019348	0.00000007	0.00027235
	PatchTST	0.00021458	0.00000009	0.00029306
	Inverted-Transformer	0.00019955	0.00000007	0.00027203
	Inverted-Informer	0.00019252	0.00000007	0.00027180
	Inverted-Flowformer	0.00020294	0.00000009	0.00029185
	Inverted-Reformer	0.00021066	0.00000009	0.00029431

5. Extended Analysis

5.1. Out-of-Sample Prediction Results Using Rolling Window

Rolling window and expanding window are two fundamental methods in forecasting. Therefore, this section further explores the prediction performance of the models under the rolling window. The rolling window length selected in this paper is 252 periods, corresponding to the number of trading days in a year. The specific results are shown in Table 2.

Table 2 presents the prediction evaluation indicators of 19 models (TsMixer, TimeMixer, CNN, TimesNet, RNN, GRU, LSTM, Informer, Autoformer, Reformer, ETSformer, FEDformer, Flowformer, Crossformer, PatchTST, Inverted-Transformer, Inverted-Informer, Inverted-Flowformer, and Inverted-Reformer) based on the rolling window. From the results, in terms of the MAE indicator, the Inverted-Informer model among Transformer-based models achieves the lowest value—it realizes a 101% improvement compared to the CNN model with the highest MAE, and a 6% improvement compared to the TsMixer model with the second lowest MAE. Regarding the MSE and RMSE indicators, the Flowformer model among Transformer-based models performs optimally. Specifically, the MSE of the Informer model is 301% higher than that of the CNN model with the highest MSE, and the MSE of the Flowformer model is 2% higher than that of the Inverted-Informer model with the second lowest MSE. It can be seen that compared with traditional neural networks; the new models improved based on classic architectures in recent years exhibit stronger prediction capabilities. However, it should be noted that this study did not perform parameter tuning for the models; all models were trained with consistent parameters, which may result in some models failing to exert their optimal performance. This issue is worthy of further exploration.

Table 2. out of sample prediction results based on rolling window

Category	Model	MAE	MSE	RMSE
MLP-based model	TsMixer	0.0002028	0.0000001	0.0002891
	TimeMixer	0.0002377	0.0000001	0.0003081
CNN-based model	CNN	0.0003860	0.0000003	0.0005156
	TimesNet	0.0002055	0.0000001	0.0002738
RNN-based model	RNN	0.0003653	0.0000002	0.0004925
	GRU	0.0003457	0.0000002	0.0004770
	LSTM	0.0003325	0.0000002	0.0004650
Transformer-based model	Informer	0.0002041	0.0000001	0.0002776
	Autoformer	0.0002472	0.0000001	0.0003331
	Reformer	0.0002721	0.0000002	0.0004244
	ETSformer	0.0002414	0.0000001	0.0003302
	FEDformer	0.0002906	0.0000002	0.0003876
	Flowformer	0.0001913	0.0000001	0.0002572
	Crossformer	0.0002281	0.0000001	0.0003284
	PatchTST	0.0002420	0.0000001	0.0003441
	Inverted-Transformer	0.0002786	0.0000002	0.0004898
	Inverted-Informer	0.0001911	0.0000001	0.0002600
	Inverted-Flowformer	0.0002233	0.0000001	0.0003446
	Inverted-Reformer	0.0002184	0.0000001	0.0003160

5.2. out of sample prediction results of long series prediction

It is noteworthy that the models improved based on MLP, CNN, and Transformer adopted in this paper are applicable not only to the short-term prediction in the previous section but also to long-term prediction. Moreover, most of the improved models in this paper were designed with long-term prediction as the priority scenario. In highly financialized markets such as the crude oil market, accurately predicting the volatility trend in the next period through models will help better grasp market information. Therefore, this section applies the improved models to long-sequence forecasting, setting pred_len (prediction length) to 22 periods (corresponding to the working days of the next month). The specific results are shown in Table 3.

Table 3 displays the long-sequence prediction evaluation indicators of the above 19 models. The results show that in terms of the MAE indicator, the Informer model among Transformer-based models performs the best. Its MAE is 114% higher than that of the CNN model with the highest MAE, and 13% higher than that of the Crossformer model with the second lowest MAE. In terms of the MSE and RMSE indicators, the Informer model among Transformer-based models also achieves the lowest values. Its MSE is 196% higher than that of the CNN model with the highest MSE, and 26% higher than that of the Autoformer model with the second lowest MSE. This again indicates that the new models improved in recent years have stronger prediction capabilities than traditional neural networks. However, limited by the consistent training parameters (without tuning), the optimal performance of the models may not be fully released, which requires in-depth research in the future.

Table 3. out of sample prediction results of long series prediction

Category	Model	MAE	MSE	RMSE
MLP-based model	TsMixer	0.00531187	0.00000241	0.00155250
	TimeMixer	0.00671615	0.00000345	0.00185835
CNN-based model	CNN	0.00854685	0.00000401	0.00200313
	TimesNet	0.00672982	0.00000384	0.00195969
RNN-based model	RNN	0.00785469	0.00000392	0.00197876
	GRU	0.00725456	0.00000357	0.00188917
	LSTM	0.00765694	0.00000386	0.00196373
Transformer-based model	Informer	0.00398983	0.00000135	0.00116327
	Autoformer	0.00457363	0.00000171	0.00130660
	Reformer	0.00563755	0.00000269	0.00164160
	ETSformer	0.00533995	0.00000211	0.00145112
	FEDformer	0.00601749	0.00000306	0.00174826
	Flowformer	0.00530821	0.00000332	0.00182214
	Crossformer	0.00452662	0.00000193	0.00138944
	PatchTST	0.00665476	0.00000380	0.00194971
	Inverted-Transformer	0.00574759	0.00000270	0.00164297
	Inverted-Informer	0.00528763	0.00000218	0.00147742
	Inverted-Flowformer	0.00567550	0.00000257	0.00160242
	Inverted-Reformer	0.00597095	0.00000287	0.00169462

6. Conclusion

The Shanghai International Energy Exchange launched China's first crude oil futures on March 26, 2018. This product has received strong support from global producers and speculators, and became the world's third-largest crude oil futures only one month after its launch, second only to WTI (West Texas Intermediate) and Brent crude oil futures. The active speculative transactions and high trading volume in the Shanghai crude oil futures market have made the importance of volatility forecasting increasingly prominent. If a reasonable model can be found to accurately predict realized volatility, it will help effectively manage and control related risks.

At the same time, with the continuous in-depth research on time series forecasting in the field of artificial intelligence, more and more models improved based on MLP, CNN, and Transformer have emerged, such as the 19 models mentioned earlier (e.g., TsMixer, TimeMixer). These models are specifically designed for time series forecasting, but their test datasets mostly come from fields such as weather, transportation, and electricity. Therefore, this paper introduces them into the prediction of crude oil realized volatility in the financial field. The results show that the improved MLP-based, CNN-based, and Transformer-based models have achieved better out-of-sample prediction results than the original models in expanding window prediction, rolling window prediction, and long-sequence prediction.

In addition, it should be noted that the training hyperparameters of all models in this study are basically consistent, which may lead to some models failing to reach the optimal accuracy. Therefore, future research can adopt hyperparameter optimization methods such as grid search and Bayesian search to select the optimal hyperparameters for each model, thereby improving prediction performance.

Regarding future research directions, it can be carried out from three aspects: First, the models used in this paper are applicable not only to univariate prediction but also to multivariate prediction. Considering the particularity of financial data, future research can introduce exogenous explanatory variables such as macroeconomic indicators, investor sentiment indicators, and technical indicators

as covariates into the model to explore whether the prediction accuracy can be further improved. Alternatively, techniques such as VMD (Variational Mode Decomposition) and CEEMDAN (Complete Ensemble Empirical Mode Decomposition with Adaptive Noise) can be used to decompose the volatility sequence, and the decomposed multivariate sequence can be input into the model for prediction. Second, the research on time series forecasting in the field of artificial intelligence continues to deepen, and new models with stronger prediction capabilities. In the future, these models can be tried for crude oil realized volatility prediction. Third, the models used in this paper can be further extended to other financial time series prediction fields such as stock returns and exchange rate fluctuations.

References

[1] Kilian L, Vigfusson R J. Do oil prices help forecast U.S. real GDP? The role of nonlinearities and asymmetries. Journal of Business and Economic Statistics, 2013, 31(1): 78-93.

[2] Aloui R, Gupta R, Miller S M. Uncertainty and crude oil returns. Energy Economics, 2016, 55: 92-100.

[3] Krane J, Medlock K B. Geopolitical dimensions of US oil security. Energy Policy, 2018, 114: 558-565.

[4] Zhang Y J, Pan X. Does the risk aversion of crude oil market investors have directional predictability for the precious metal and agricultural markets? China Agricultural Economic Review, 2021, 13(4): 894-911.

[5] Li Y, Jiang S, Li X, et al. The role of news sentiment in oil futures returns and volatility forecasting: Data-decomposition based deep learning approach. Energy Economics, 2021, 95.

[6] Brandt M W, Gao L. Macro fundamentals or geopolitical events? A textual analysis of news events for crude oil. Journal of Empirical Finance, 2019, 51.

[7] Zhang Y J, Yan X X. The impact of US economic policy uncertainty on WTI crude oil returns in different time and frequency domains. International Review of Economics and Finance, 2020, 69.

[8] Jackson J K, Weiss M A, Schwarzenberg A B, et al. Global economic effects of COVID-19M//The Effects of COVID-19 on the Global and Domestic Economy. 2021.

[9] Qureshi A, Rizwan M S, Ahmad G, et al. Russia–Ukraine war and systemic risk: Who is taking the heat? Finance Research Letters, 2022, 48: 103036.

[10] Qin X. Oil shocks and financial systemic stress: International evidence. Energy Economics, 2020, 92: 104945.

[11] Orhan E. THE EFFECTS OF THE RUSSIA-UKRAINE WAR ON GLOBAL TRADE//Journal of International Trade, Logistics and Law: Vol. 8. 2022.

[12] Vaswani A, Shazeer N, Parmar N, et al. Attention is All you Need//GUYON I, VON LUXBURG U, BENGIO S, et al. Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA. 2017: 5998-6008.

[13] Brown T B, Mann B, Ryder N, et al. Language Models are Few-Shot Learners//LAROCHELLE H, RANZATO M, HADSELL R, et al. Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual. 2020: 1877-1901.

[14] Dosovitskiy A, Beyer L, Kolesnikov A, et al. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale//9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021. OpenReview.net, 2021.

[15] Zhou H, Zhang S, Peng J, et al. Informer: Beyond Efficient Transformer for Long Sequence Time-Series Forecasting//35th AAAI Conference on Artificial Intelligence, AAAI 2021: Vol. 12B. 2021: 11106-11115.

[16] Wu H, Xu J, Wang J, et al. Autoformer: Decomposition Transformers with Auto-Correlation for Long-Term Series Forecasting//RANZATO M, BEYGELZIMER A, DAUPHIN Y N, et al. Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual. 2021: 22419-22430.

[17] Zeng A, Chen M, Zhang L, et al. Are Transformers Effective for Time Series Forecasting?//WILLIAMS B, CHEN Y, NEVILLE J. Thirty-Seventh {AAAI} Conference on Artificial Intelligence, {AAAI} 2023, Washington, DC, USA, February 7-14, 2023: Vol. 37. {AAAI} Press, 2023: 11121-11128[2025-04-24].

[18] Liu Y, Hu T, Zhang H, et al. iTransformer: Inverted Transformers Are Effective for Time Series Forecasting//The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024. OpenReview.net, 2024.

[19] Chen S A, Li C L, Yoder N, et al. TSMixer: An all-MLP Architecture for Time Series Forecasting. CoRR, 2023, abs/2303.06053.

[20] Wang S, Wu H, Shi X, et al. TimeMixer: Decomposable Multiscale Mixing for Time Series Forecasting//The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024. OpenReview.net, 2024.

[21] Kitaev N, Kaiser L, Levskaya A. Reformer: The Efficient Transformer//8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020. OpenReview.net, 2020.

[22] Woo G, Liu C, Sahoo D, et al. ETSformer: Exponential Smoothing Transformers for Time-series Forecasting. CoRR, 2022, abs/2202.01381.

[23] Zhou T, Ma Z, Wen Q, et al. FEDformer: Frequency Enhanced Decomposed Transformer for Long-term Series Forecasting//CHAUDHURI K, JEGELKA S, SONG L, et al. International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA: Vol. 162. PMLR, 2022: 27268-27286.

[24] Wu H, Wu J, Xu J, et al. Flowformer: Linearizing Transformers with Conservation Flows//CHAUDHURI K, JEGELKA S, SONG L, et al. International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA: Vol. 162. PMLR, 2022: 24226-24242.

[25] Zhang Y, Yan J. Crossformer: Transformer Utilizing Cross-Dimension Dependency for Multivariate Time Series Forecasting//The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023. OpenReview.net, 2023.

[26] Nie Y, Nguyen N H, Sinthong P, et al. A Time Series is Worth 64 Words: Long-term Forecasting with Transformers//The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023. OpenReview.net, 2023.

[27] Wu H, Hu T, Liu Y, et al. TimesNet: Temporal 2D-Variation Modeling for General Time Series Analysis//The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023. OpenReview.net, 2023.